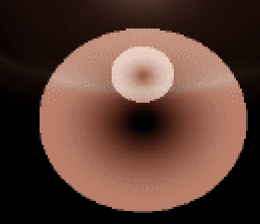
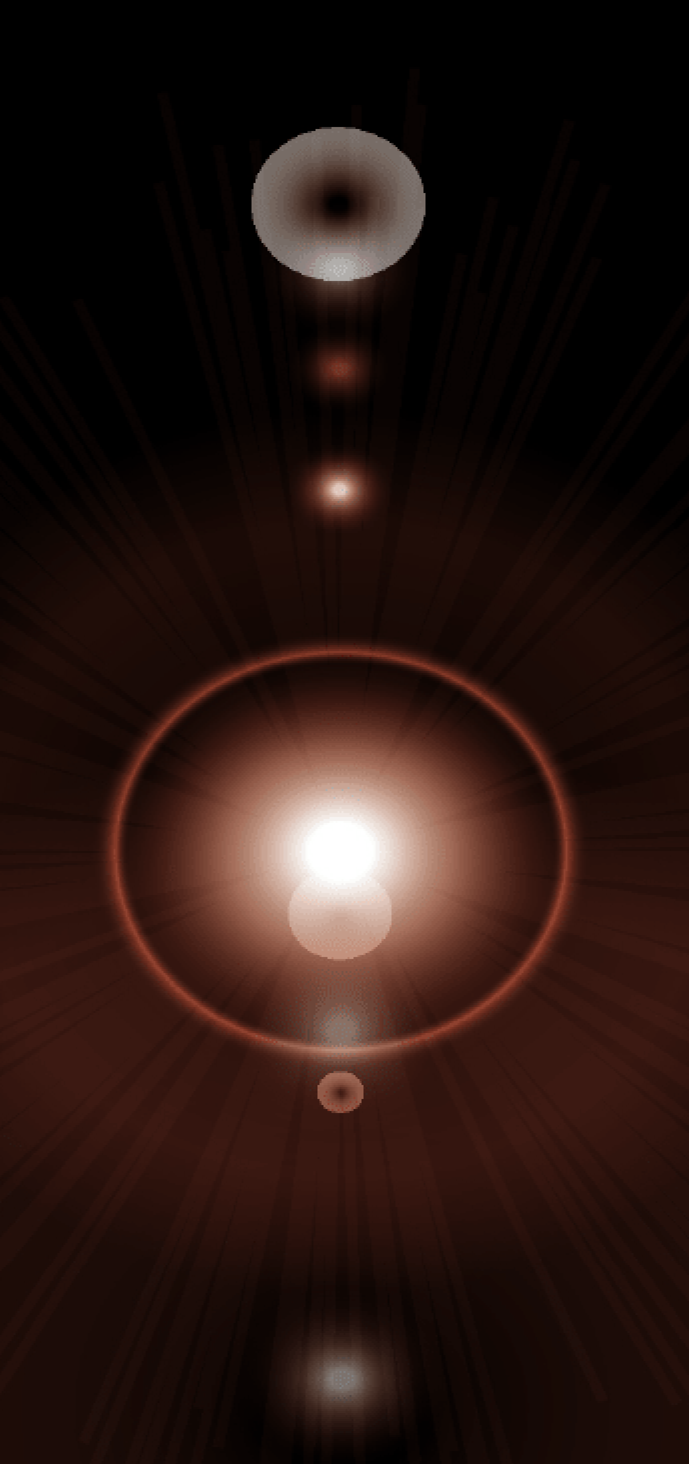


普通高等教育"十一五"国家级规划教材



陈　明 编著

软件测试技术



21世纪计算机科学与技术实践型教程

丛书主编 陈明

清华大学出版社

21世纪计算机科学与技术实践型教程

软件测试技术

陈　明　编著

清华大学出版社
北　京

内 容 简 介

本书是计算机软件测试课程教材，主要内容包括软件测试概述、软件测试方法、软件测试过程、面向对象测试、测试的设计与实现、Web测试、软件测试自动化、软件质量与质量保证、软件测试工具等内容。

本书可作为高等学校计算机科学与技术专业的软件测试课程教材，也可作为计算机软件开发人员的参考书。

图书在版编目（CIP）数据

软件测试技术/陈明编著．—北京：清华大学出版社，2011.2
（21世纪计算机科学与技术实践型教程）
ISBN 978-7-302-23780-8

Ⅰ．①软…　Ⅱ．①陈…　Ⅲ．①软件—测试—高等学校—教材　Ⅳ．①TP311.5

中国版本图书馆CIP数据核字(2010)第171220号

责任编辑：谢　琛　王冰飞
责任校对：李建庄
责任印制：王秀菊

出版发行：清华大学出版社　　**地　　址：**北京清华大学学研大厦A座
http://www.tup.com.cn　　**邮　　编：**100084
社　总　机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62795954，jsjjc@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印　装　者：北京密云胶印厂
经　　销：全国新华书店
开　　本：185×260　　**印　　张：**12.75　　**字　　数：**312千字
版　　次：2011年2月第1版　　**印　　次：**2011年2月第1次印刷
印　　数：1～4000
定　　价：25.00元

产品编号：039513-01

《21世纪计算机科学与技术实践型教程》

编辑委员会

《21 世纪计算机科学与技术实践型教程》

序

21 世纪影响世界的三大关键技术：以计算机和网络为代表的信息技术；以基因工程为代表的生命科学和生物技术；以纳米技术为代表的新型材料技术。信息技术居三大关键技术之首。国民经济的发展采取信息化带动现代化的方针，要求在所有领域中迅速推广信息技术，导致需要大量的计算机科学与技术领域的优秀人才。

计算机科学与技术的广泛应用是计算机学科发展的原动力，计算机科学是一门应用科学。因此，计算机学科的优秀人才不仅应具有坚实的科学理论基础，而且更重要的是能将理论与实践相结合，并具有解决实际问题的能力。培养计算机科学与技术的优秀人才是社会的需要、国民经济发展的需要。

制定科学的教学计划对于培养计算机科学与技术人才十分重要，而教材的选择是实施教学计划的一个重要组成部分，《21 世纪计算机科学与技术实践型教程》主要考虑了下述两方面。

一方面，高等学校的计算机科学与技术专业的学生，在学习了基本的必修课和部分选修课程之后，立刻进行计算机应用系统的软件和硬件开发与应用尚存在一些困难，而《21 世纪计算机科学与技术实践型教程》就是为了填补这部分空白。将理论与实际联系起来，使学生不仅学会了计算机科学理论，而且也学会应用这些理论解决实际问题。

另一方面，计算机科学与技术专业的课程内容需要经过实践练习，才能深刻理解和掌握。因此，本套教材增强了实践性、应用性和可理解性，并在体例上做了改进——使用案例说明。

实践型教学占有重要的位置，不仅体现了理论和实践紧密结合的学科特征，而且对于提高学生的综合素质，培养学生的创新精神与实践能力有特殊的作用。因此，研究和撰写实践型教材是必需的，也是十分重要的任务。优秀的教材是保证高水平教学的重要因素，选择水平高、内容新、实践性强的教材可以促进课堂教学质量的快速提升。在教学中，应用实践型教材可以增强学生的认知能力、创新能力、实践能力以及团队协作和交流表达能力。

实践型教材应由教学经验丰富、实际应用经验丰富的教师撰写。此系列教材的作者不但从事多年的计算机教学，而且参加并完成了多项计算机类的科研项目，他们把积累的经验、知识、智慧、素质融合于教材中，奉献给计算机科学与技术的教学。

我们在组织本系列教材过程中，虽然经过了详细的思考和讨论，但毕竟是初步的尝试，不完善甚至缺陷不可避免，敬请读者指正。

本系列教材主编　陈明

2005 年 1 月于北京

前　　言

计算机软件是逻辑产品。软件与硬件具有完全不同的特征。计算机软件现已成为一种新的驱动力，是进行决策的引擎，是现代工程研究和解决问题的基础。在各种类型的应用系统中无所不在，具有十分广泛的应用。

随着软件企业规模的扩大，复杂度的不断提高，软件测试难度也进一步加大，凸显了软件测试的重要性。软件测试是软件工程学科的重要分支，现已成为软件质量保证的关键技术之一。在软件开发过程中，软件测试是不可缺少的重要环节，软件测试工作直接决定了软件产品的质量。

软件测试工具是支持软件生存周期中某一阶段的测试任务实现而使用的计算机程序。软件测试环境是一组相关的软件测试工具的集合，将它们集成在一起支持软件测试。软件测试工具与环境是软件测试的重要组成部分，对于提高软件生产率，改进软件质量有越来越大的作用。

软件测试是异常活跃的技术，需要丰富的想象力。软件测试又是一个实践性极强的实用技术，在学习中，既要学习基本的理论知识，又要掌握必要的技能，也就是说，不仅要能掌握其理论原则与方法，更重要的是能熟练地进行应用。软件测试人才的需要日益增多，通过软件测试的理论学习与实践，可以培养学生掌握测试的基本内容和方法，并在软件开发的工作中得以贯彻，进而展现学科的力量。

在学习软件测试技术过程中，要注重技术的应用，通过大量的时间和思考，理解软件测试的思想和理念，并运用测试技术和技巧去解决问题。

全书分为9章，主要包括软件测试概述、软件测试方法、软件测试过程、面向对象测试、测试的设计与实现、Web测试、软件测试自动化、软件质量与质量保证、软件测试工具等内容。

在内容选择上，注重先进与系统；在结构上，各章呈模块化。在描述中，面向实践，注重理论与实践的结合，有助于快速掌握软件测试必需的技术和方法，促进软件测试能力的培养。

由于作者水平有限，书中不足之处在所难免，敬请批评指正。

陳明

2010年12月于北京

目　　录

第 1 章　概　　述

学习要点：

❖ 软件缺陷。

❖ 软件测试的定义。

❖ 软件测试的对象。

❖ 软件测试的目的。

计算机科学技术的飞速发展，促进了软件产品的广泛应用，不论是软件的生产者还是使用者，都在激烈的竞争中求生存，软件产品的质量已经成为关注的焦点。软件开发者为了占有市场，必须把产品质量作为企业的重要目标之一，进而才可以确保在激烈的竞争中获得胜利。为了保证软件产品的质量，软件测试成为必不可少的重要过程与手段。

在开发大型软件系统的过程中，面对极其错综复杂的问题，软件开发者的主观认识不可能完全符合客观事实，而且与工程密切相关的各类人员之间的沟通和配合也不可能完美无缺，因此，在软件生存周期的每个阶段不可避免地产生差错。尽管力求在每个阶段结束之前通过严格的技术审查，尽可能早地发现并纠正差错，但是，审查并不能发现所有错误，此外，在这些错误存在的过程中还可能引入新的错误。如果软件在投入实际运行之前，没有发现并纠正它存在的部分错误，那么这些错误将在运行过程中暴露出来，不仅要为改正这些错误而付出高昂的代价，而且很可能产生严重后果。

软件测试在软件生存周期中经历两个阶段。在编写出每个模块之后就对它做测试，称为单元测试，模块的编写者和测试者是同一个人，编码和单元测试在软件生存周期中属于同一个阶段。在这个阶段结束之后，对软件系统还应该进行各种综合测试，这是软件生存周期中的另一个独立的阶段，由专门的测试人员承担这项工作。

大量统计结果表明，软件测试的工作量占软件开发总工作量的 40%以上，在特殊情况下，例如对关系到人的生命安全的软件要进行的测试所花费的成本，可能相当于软件工程其他开发步骤总成本的 3～5 倍。因此，必须重视软件测试，绝不要以为编写出程序之后软件开发工作就完成了，实际上，几乎还需要完成与开发工作同样多的工作量。

1.1　软件测试的发展

随着社会化生产，应运而生的测试技术涉及多方面。在许多领域，测试都是保证产品质量的关键。软件测试是软件工程的一部分重要内容。

随着计算机的产生与发展,软件开发和软件测试相继出现。由于早期的计算机性能比较差,软件的可编程范围也比较狭窄,在这一阶段并没有系统的软件测试,更多的是一种调试性测试,测试主要是为了证明系统的可运行性。

20 世纪 50 年代后期到 20 世纪 60 年代,许多高级语言相继诞生并且得到了广泛的应用,测试的对象逐渐转入到用高级语言书写的系统。但是,由于受到硬件系统发展瓶颈的限制,软件测试位于次要地位,软件的正确性和可用性主要由编程人员的水平所决定。因此,软件测试理论和方法的发展缓慢。

20 世纪 70 年代以后,随着计算机处理速度的提高,软件在整个系统中的重要性变得越来越重要。一方面在这个阶段,软件的规模越来越大,可视化的编程环境、日益完善的软件分析设计方法以及新的软件开发过程模型的出现使得大型软件的开发成为可能;另一方面,由于软件规模和复杂性的迅速增加,软件面临着巨大的危机,软件测试得以重视并提到日程。

20 世纪 70 年代中期,软件测试技术的研究达到高潮。J. B. Goodenough 和 S. L. Gerhart 首先提出了软件测试的理论,从而把软件测试这一实践性很强的学科提高到理论的高度。1982 年 6 月在美国北卡罗来纳大学召开了首次软件测试的技术会议,讨论了软件测试问题,这次会议是软件测试技术发展中的一个重要里程碑。

软件产业的发展,对软件的成本、进度和质量都提出了更高的要求,对软件质量的控制已不再是传统意义上的软件测试。传统的测试一般在软件开发后期才介入,然而,大量研究结果表明设计活动引入的错误占软件开发过程中出现的所有错误的 50%～65%。因此,测试就已经不再是一个编码后才进行的活动,而是一个基于软件开发整个生存周期的质量控制活动。

目前,在软件测试理论、测试方法、测试过程和测试工具等方面的研究取得了大量的进展。这不仅使得软件的质量有了基本的保证,也使得软件测试的工作量占到了软件开发总工作量的 40%以上,软件测试的地位上升到前所未有的高度。

1.2 软件错误与软件缺陷

1.2.1 软件错误与缺陷的概念

1. 软件错误

在编写代码时有可能会出现错误,把这种错误叫做 Bug。错误在整个软件开发周期中很可能扩散,在需求阶段发生的错误,在设计期间有可能被放大,在编写代码时还会进一步扩大。

2. 软件缺陷

缺陷是错误的结果。更精确地说,缺陷是错误的表现,缺陷很难捕获。当设计人员出现遗漏错误时,所导致的缺陷会是遗漏本来应该在表现中提供的内容。这种情况表明需要对定义做进一步的细化。把缺陷分为错误缺陷和遗漏缺陷。如果把某些信息输入到不

正确的表示中，就是错误缺陷；如果在设计过程中没有输入某些正确且必要的信息，就是遗漏缺陷。在这两类缺陷中，后者更难检测和解决。

1.2.2　软件错误类型及出现的原因

1. 软件错误类型

根据软件错误的性质不同，可以把软件错误分为下述几种类型。

1）需求错误

软件需求指定的不合理或不正确；需求不完全；需求中含有逻辑错误；需求分析的文档有误，等等。

2）功能与性能错误

功能或性能存在错误，或是遗漏了某些功能，或是规定了某些冗余的功能；为用户提供的信息有错，或信息不确切；对异常情况处理有误，等等。

3）软件结构错误

程序控制流或控制顺序有误；处理过程有误，等等。

4）数据错误

数据定义或数据结构有误；数据存取或数据操作有误，等等。例如：动态数据与静态数据混淆、参数与控制数据混淆等。

5）实现和编码错误

编码错误包括语法错误、数据名错误、局部变量与全局变量混淆或者程序逻辑有误等。

6）集成错误

软件的内部接口、外部接口有误；软件各相关部分在时间配合、数据吞吐量等方面不协调，等等。

7）系统结构错误

操作系统调用错误或使用错误、恢复错误、诊断错误、分割及覆盖错误以及引用环境的错误等。

8）测试定义与测试执行错误

测试的错误包括测试方案设计与测试实施的错误、测试文档的问题、测试用例不够充分等。普遍出现的软件结构错误、数据错误和功能与性能错误特别受到重视。

2. 出现错误的原因

软件出现错误的原因是多方面的，归纳起来主要有如下几点。

(1) 交流不够、交流上有误解或者根本没有进行交流。在不清晰应该做什么或不应该做什么的情况下进行了应用开发。

(2) 软件复杂性。图形用户界面(GUI，Graphic User Interface)，客户/服务器结构，分布式应用，数据通信，超大型关系型数据库以及庞大的系统规模，使得软件复杂性呈指数增长。

(3) 程序设计错误。在软件设计阶段出现的错误，主要包括概要设计、详细设计和编码步骤出现了错误。

(4) 需求不断变化。需求变化的后果可能是造成系统重新设计、项目日程的重新安排、已经完成的工作可能要部分重做甚至完全抛弃等。如果有许多小的改变或者一次大的变化，项目各部分之间已知或未知的关系相互影响进而导致更多问题的出现，还可能影响工程参与者的积极性。

(5) 时间压力。软件项目的日程表很难做到准确，很多时候需要预计和猜测。当最终期限到来之际，由于时间紧迫，容易出现错误。

(6) 代码文档不完全。在一些团队中，不鼓励其程序员为代码编写文档，也不鼓励程序员将代码写得清晰和容易理解，相反认为少写文档可以更快地进行编码，无法理解的代码更易于工作的保密，显然，这是一种错误的认识。

(7) 软件开发工具。当软件产品的开发依赖于某些工具时，那么这些工具本身隐藏的问题可能会导致产品的错误。因此，应该选择比较成熟的开发工具，而不是追求最先进的开发工具。

1.2.3 软件缺陷的主要特征

软件错误有多种类型，在一些关键系统中，出现软件错误时，其后果是灾难性的。而在非关键性系统中，出现错误的后果可能并不像前一种情况那样明显，但难以观察。在通常情况下，利用软件缺陷描述软件错误。软件缺陷的主要特征如下。

(1) 软件未达到软件产品需求说明书指明的要求。

(2) 软件出现了软件产品需求说明书中指明不应出现的错误。

(3) 软件功能超出软件产品需求说明书指明的范围。

(4) 软件未达到软件产品需求说明书未指明但应达到的要求。

(5) 软件测试人员认为难以理解、不易使用、运行速度慢或最终用户认为不好。

考虑到设计等方面的因素，软件缺陷还包括软件设计不符合规范，未能在特定的条件下(资金、范围等)达到最佳等。但是，更多的是把软件缺陷看成软件运行时出现的各种问题。

统计结果表明：大多数软件缺陷并非源自编码错误，导致软件缺陷的最大原因是需求分析错误，其次是设计错误，还有编码错误和测试错误等。

1.3 软件测试的定义

软件测试的定义是：软件测试是为了发现错误而执行程序的过程。这个定义明确指出寻找错误是测试的目的。

软件测试是软件工程过程的一个重要阶段，在软件投入运行前，对软件需求分析、设计和编码各阶段产品的最终检查，是为了保证软件开发产品的正确性、完全性和一致性，从而进行检测错误以及修正错误的过程。软件开发的目的是开发出满足用户需求的高质

量、高性能的软件产品，而软件测试以检查软件产品内容和功能特性为核心，是软件质量保证的关键步骤，也是成功实现软件开发目标的重要保障。

从用户的角度来看，普遍希望通过软件测试找出软件中隐藏的错误，所以软件测试应该是为了发现错误而执行程序的过程。软件测试应该根据软件开发各阶段的规格说明和程序的内部结构而精心设计一批测试用例（即输入数据及其预期的输出结果），并利用这些测试用例去运行程序，以发现程序中隐藏的错误。

软件测试的主要作用如下。

（1）测试是执行一个系统或者程序的操作。

（2）测试是带着发现问题和错误的意图来分析和执行程序。

（3）测试结果可以检验程序的功能和质量。

（4）测试可以评估项目产品是否获得预期目标和可以被客户接受的结果。

（5）测试不仅包括执行代码，还包括对需求等编码以外的测试。

1.4　软件测试的对象

软件开发期间任何一个环节发生了问题都可以在软件测试中表现出来。软件测试应该贯穿软件开发期间。因此需求分析、概要设计、详细设计以及程序编码等各阶段所得到的文档，包括需求规格说明、概要设计规格说明、详细设计规格说明以及源程序，都是软件测试的对象。

1.5　软件测试的目的

软件测试的目的就是找出被测试软件所有存在的错误，但实际上，测试人员不可能发现所有的错误。一个不成功的测试是没有找到错误的测试，成功的测试是花费最少的时间和人力找出软件中潜在的各种错误。

如果根据上述目标成功构造了测试，则能够揭示软件中存在的错误。测试能证实软件根据需求所具有的功能和性能，此外，在构造测试方案的过程中，收集的数据可以为软件可靠性以及软件的整体质量提供一些比较重要的信息。但是，测试无法说明错误不存在，只能揭示目前的软件的状态良好。

软件测试不以发现错误为唯一目的，查不出错误的测试并非没有价值。通过分析错误产生的原因和错误的分布特征，可以帮助发现当前所采用的软件过程中的缺陷并加以改进。同时，这种分析也能帮助设计出有针对性的检测方法，改善测试的有效性。没有发现软件中错误的测试也是有价值的，因为整个测试过程本身就是评定测试软件质量的一种方法。如果在运行多次后或者重新构建一套测试软件后而仍未发现软件错误，这样可以得出这样的结论：被测试软件已经比较完美。因为存在不同的针对性，所以软件测试也就存在多种目的，其中最重要的三条如下。

(1) 证明测试人员所做的是客户所需的。

(2) 确保编程人员正确理解设计的意图。

(3) 通过回归测试来保证目前运行的程序在将来仍然可以正常工作。

测试目的决定了测试方案的设计。如果为了表明程序是正确的而进行测试,就很可能设计一些不易暴露错误的测试方案;相反,如果测试是为了发现程序中存在的错误,就会设计出最能暴露错误的测试方案。

1.6 软件测试的原则

经过理论分析和工作实践,总结如下一些软件测试原则。

1. 尽早不断测试的原则

应当尽早不断地进行软件测试。据统计,约60%的错误来自设计以前,并且修正一个软件错误所需的费用将随着软件生存周期的进展而上升。错误发现得越早,修正它所需的费用就越少。

2. IPO原则

测试用例由测试输入数据和与之对应的预期输出结果这两部分组成。

3. 独立测试原则

独立测试原则是指软件测试工作由在经济上和管理上独立于开发机构的组织进行。程序员应避免检查自己的程序,程序设计机构也不应测试自己开发的程序。软件开发者难以客观、有效地测试自己的软件,而找出那些因为对需求的误解而产生的错误就更加困难。采用独立测试原则的优点如下所述。

(1) 客观性:经济上的独立性使其工作有更充分的条件按测试要求去完成。

(2) 专业性:软件测试需要有专业队伍加以研究,并进行工程实践。专业化分工是提高测试水平、保证测试质量、充分发挥测试效率的必然途径。

(3) 权威性:由于专业优势,独立测试工作形成的测试结果更具有信服力和权威性。

4. 合法和非合法原则

在设计时,测试用例应当包括合法的输入条件和不合法的输入条件。

5. 错误群集原则

软件错误呈现群集现象。经验表明,某程序段剩余的错误数目与该程序段中已发现的错误数目成正比,所以应该对错误群集的程序段进行重点测试。

6. 严格性原则

严格执行测试计划,排除测试的随意性。

测试计划应包括所测软件的功能,输入和输出,测试内容,各项测试的进度安排,资源要求,测试资料,测试工具,测试用例的选择,测试的控制方法和过程,系统的组装方式,跟踪规则,调试规则,回归测试的规定以及评价标准等。

7. 覆盖原则

应当对每一个测试结果做全面的检查。

8. 定义功能测试原则

检查程序是否做了要做的事仅是成功的一半，另一半是看程序是否做了不属于它做的事。

9. 回归测试原则

应妥善保留测试用例，不仅可以用于回归测试，也可以为以后的测试提供参考。

10. 错误不可避免原则

在测试时不能首先假设程序中没有错误。

1.7 软件测试的重要性

20 世纪 70 年代以后，逐渐形成了软件生存期的概念，而且对于软件产品的质量保障以及组织对软件开发有着重要的意义。

软件测试可以保证对需求和设计的理解与表达的正确性、实现的正确性以及运行的正确性。因为任何一个环节发生了问题都将在软件测试中表现出来，同时测试还可防止由于无意识的行为而引入的一些可能出现的错误。例如对一些功能进行更改、分解或者扩展时而不小心引入的错误，也许就会对整个程序功能造成不可预料的破坏。所以一旦发生了上面的情况，就需要对更新后的工作结果进行重新测试。如果测试没有通过，则说明某个环节出现了问题。

软件测试是软件质量保证的重要手段。在软件开发总成本中，测试上的开销要占到30%～50%。如果把维护阶段也考虑在内，整个软件生存期中，开发测试的成本比例会有所降低，但维护工作相当于二次开发，乃至多次开发，其中也包含有许多测试工作。因此，估计软件工作有 50%的时间和 50%以上的成本花在测试工作上。由此可见，要成功开发出高质量的软件产品，必须重视并加强软件测试工作。归纳起来，软件测试的重要性表现如下。

(1) 一个不好的测试程序可能导致任务的失败，更严重的是可能影响操作的性能和可靠性，并且可能导致在维护阶段花费巨大的成本。

(2) 一个好的测试程序是项目的主要保证。复杂的项目在软件测试和验证上需要花费超过项目一半以上的成本。为了使测试有效，必须事先在计划和组织测试上面花费适当的时间。

(3) 一个好的测试程序可以极大地帮助定义需求和设计。这有助于项目在一开始就步入正轨，测试程序的好坏对整个项目的成功有着重要的影响。

(4) 一个好的测试可以使修改错误的成本变得很低。

(5) 一个好的测试可以弥补一个不好的软件项目，有助于发现项目存在的许多问题。

1.8 软件测试的复杂性

只有穷举测试才能找出软件系统中所有的错误，但穷举测试具有复杂性，所以软件测试具有复杂性。彻底测试就是让被测程序在一切可能的输入情况下全部执行一遍，通常也称这种测试为穷举测试。黑盒法是穷举输入测试，只有把所有可能的输入都作为测试情况使用，才能以这种方法查出程序中所有的错误。实际上测试情况有无穷多个，不仅要测试所有合法的输入，而且还要对那些不合法但是可能的输入进行测试，显然不可能。白盒法是穷举路径测试，贯穿程序的独立路径数是天文数字，但即使每条路径都测试了仍然可能有错误，主要有如下一些原因。

(1) 穷举路径测试不能查出程序是错误的程序。

(2) 穷举路径测试不能查出程序中遗漏路径的错误。

(3) 穷举路径测试可能发现不了一些与数据相关的错误。

1.9 软件测试的经济性

程序测试只能证明错误的存在，但不能证明错误不存在。在实际测试中，穷举测试工作量太大，在实践上行不通，这就表明一切实际测试都是不彻底的。当然就不能够保证被测试程序中不存在遗留的错误。软件工程的总目标是充分利用有限的人力和物力资源，高效率、高质量地完成测试。为了降低测试成本，选择测试用例时应注意经济性的原则。

(1) 根据程序的重要性和一旦发生故障将造成的损失来确定它的测试等级。

(2) 研究测试策略，以便能使用尽可能少的测试用例，发现尽可能多的程序错误。掌握好测试量是至关重要的。测试不充分意味着让用户承担隐藏错误带来的危险，而过度测试则会浪费许多宝贵的资源。

测试是软件生存期中费用消耗最大的环节。测试费用除了测试的直接消耗外，还包括其他的相关费用。能够决定需要做多少次测试的主要影响因素如下。

1) 系统的目的

系统的目的影响所需要进行的测试的数量。那些可能产生严重后果的系统必须要进行更多的测试。一个用来控制密封燃气管道的系统应该比一个与有毒爆炸物品无关的系统有更高的可信度。一个安全关键软件的开发组比一个游戏软件开发组有要严格得多的查找错误方面的要求。

2) 潜在的用户数量

一个系统的潜在用户数量也在很大程度上影响了测试必要性的程度。一个在全世界范围内有几千个用户的系统肯定比一个只在办公室中运行的有两三个用户的系统需要更多的测试。除此而外，如果在内部系统中发现了一个严重的错误，在处理错误的时候的费用就相对少一些，如果要处理一个遍布全世界的错误就需要花费相当大的财力和精力。

3) 信息的价值

在考虑测试的必要性时，需要将系统的价值考虑在内，一个支持许多家大银行或众多

证券交易所的客户/服务器系统比一个支持小商店的系统要进行更多的测试。这两个系统的用户都希望得到高质量、无错误的系统，但是前一种系统的影响力比后一种要大得多。因此应该从经济方面考虑，投入与经济价值相对应的时间和费用去进行测试。

4）开发机构

一个没有标准和缺少经验的开发机构很可能开发出存在较多错误的系统。而一个建立了严格标准和有很多经验的开发机构开发出来的系统存在的错误就相对少一些，因此，对于不同的开发机构来说，所需要的测试必要性也不同。

5）测试的时机

测试量随着时间的推移发生改变。在一个竞争激烈的市场里，争取时间可能是制胜的关键，开始可能不在测试上花太多时间，但以后市场分配格局建立起来了，那么产品的质量就变得更重要了，测试量就要加大。也就是说，测试量应该针对合适的目标进行调整。

1.10 开发各阶段的测试

软件开发过程是一个自顶向下、逐步细化的过程，首先在软件计划阶段定义了软件的作用域，然后进行软件需求分析，建立软件的数据域、功能和非功能需求、约束和一些有效性准则。接着进入软件开发阶段，首先是软件设计，然后再把设计用某种程序设计语言转换成程序代码表现出来。而测试过程则是按相反的顺序安排的自底向上、逐步集成的过程。低一级测试为上一级测试准备条件，进行确认测试。两者也可以进行平行的测试。

进行软件测试，首先要对每一个程序模块进行单元测试，消除程序模块内部在逻辑上和功能上的错误。其次对照软件设计进行集成测试，检测和排除子系统在系统结构上的错误。随后再对照需求，进行确认测试。最后从系统全体出发，运行系统，检查是否满足要求。软件测试与软件开发过程的关系如图 1-1 所示。

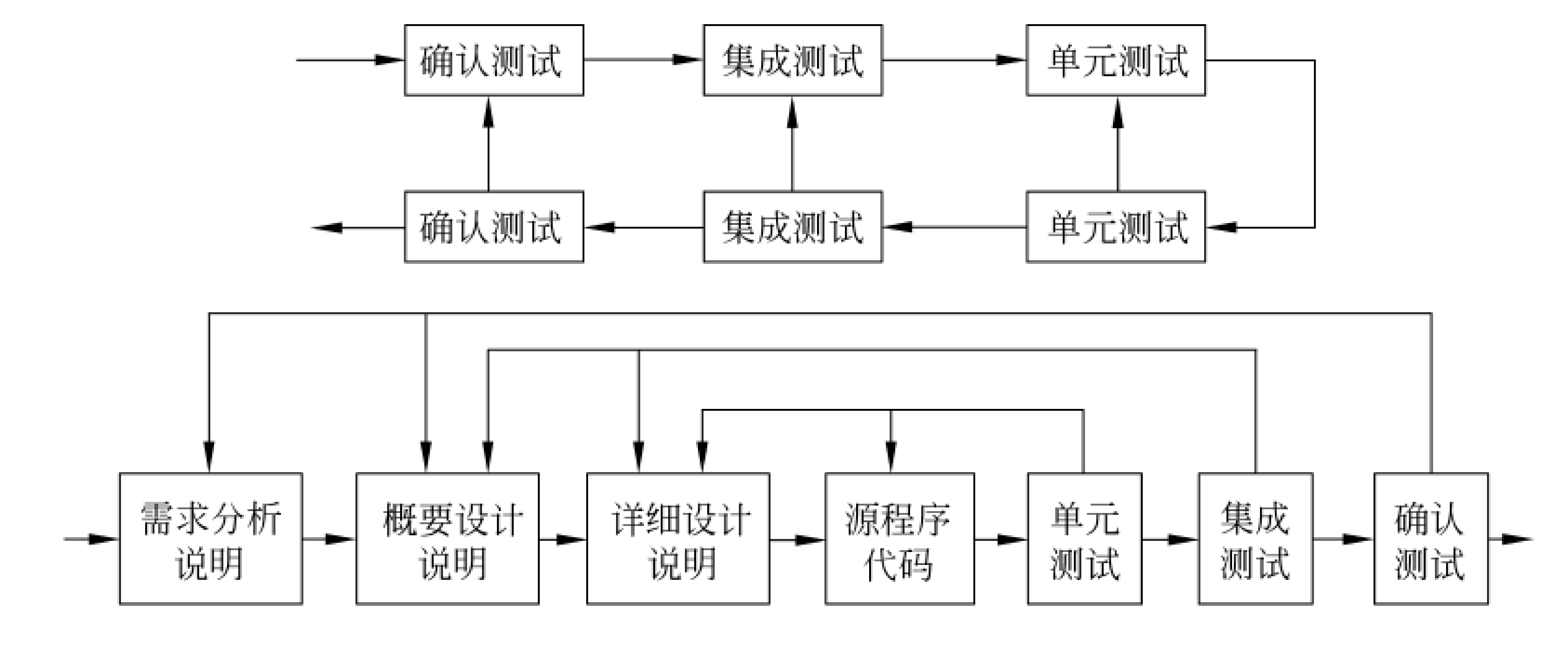


图 1-1 软件测试与软件开发过程的关系

小 结

随着软件危机的出现以及人们对软件质量的进一步认识，软件测试的地位得到了前所未有的提高。测试已经不再是软件开发中的一个阶段，而是贯穿了软件开发的整个过

程。软件测试是对软件规格说明、设计和编码强化评审和审查，是保证软件质量的关键步骤，软件测试可以降低软件开发成本。软件测试的目的是尽可能多地发现软件中的错误，通过测试发现问题，并进行问题解决，再进行回归测试，从而提高软件质量。

在软件开发过程中，测试开始的时间越早，测试执行的频率越高，整个软件开发成本降低得就越多。另外，随着软件开发规模的扩大、复杂程度的增加，软件测试工作也越来越重要，其工作难度也越来越大。

本章主要介绍了软件测试的一些基本概念，包括软件测试的发展、定义、对象、目的、原则、重要性以及软件错误的表现和原因。这些内容为掌握软件测试技术建立了基本基础。

习 题 1

1. 软件测试的对象包括(　　)。

A. 目标程序和相关文档　　B. 源程序、目标程序、数据及相关文档

C. 目标程序、操作系统和平台软件　　D. 源程序和目标程序

2. 某软件公司在招聘软件测试员时，应聘者甲向公司作如下保证：

(1) 经过自己测试的软件今后不会再出现问题；

(2) 在工作中对所有程序员一视同仁，不会因为在某个程序员编写的程序中发现的问题多，就重点审查该程序，以免不利于团结；

(3) 承诺不需要其他人员，自己就可以独立地进行测试工作；

(4) 发扬咬定青山不放松的精神，不把问题都找出来，决不罢休。

认为应聘者甲的保证(　　)。

A. (1)、(4)是正确的　　B. (2)是正确的

C. 都是正确的　　D. 都不正确

3. 软件测试的目的观点如下：

(1) 度量与评估软件的质量；

(2) 保证软件质量；

(3) 改进软件开发过程；

(4) 发现软件错误。

其中正确的是(　　)。

A. (1)、(2)、(3)　　B. (1)、(2)、(4)

C. (1)、(3)、(4)　　D. (1)、(2)、(3)、(4)

4. 下列是对软件测试的一些认识，哪些是不正确的？(　　)

A. 软件开发完成后进行软件测试

B. 软件发布后如果发现质量问题，那是软件测试人员的错

C. 软件测试不仅是测试人员的事情，而且也与程序员有关

D. 软件测试是没有前途的工作，只有程序员才是软件高手

第 2 章　软件测试方法

学习要点：

❖ 黑盒测试。
❖ 等价类划分。
❖ 边界值划分。
❖ 白盒测试。
❖ 程序结构分析。
❖ 白盒测试和黑盒测试的比较。
❖ 敏捷测试。

2.1　静态分析

静态分析是不需要执行程序而进行测试的技术，其主要功能是检查软件与其描述是否一致，是否有冲突或是歧义性。静态分析的主要特征是分析源程序，而不是运行程序。它检查的是软件系统在描述、表示和规格上的错误。静态分析是其他测试的前提，静态分析包括代码检查、静态结构分析、代码质量度量等。静态分析可由人工进行，充分发挥人的逻辑思维优势，也可借助软件工具进行，加快分析速度和效果。

1. 代码检查

代码检查包括代码走查和代码审查，代码走查与代码审查的区别是：代码走查除了阅读程序外，还需要由测试员利用人工运行程序并得出输出结果，然后由参加者对结果进行审查，以达到测试的目的。代码审查的主要内容是检查代码和设计的一致性，代码对标准的遵循程度、可读性，代码逻辑表达的正确性，代码结构的合理性等方面；代码审查可以发现违背程序编写标准的问题，程序中不安全、不明确和模糊的部分，找出程序中不可移植部分、违背编程风格的问题，包括变量检查、命名和类型审查、程序逻辑审查、程序语法检查和程序结构检查等内容。

2. 静态结构分析

静态结构分析主要是以图形的方式描述程序的内部结构，例如函数调用关系图、函数内部控制流图。其中函数调用关系图以图形方式描述一个应用程序中各个函数的调用和被调用关系；函数内部控制流图显示一个函数的逻辑结构，它由许多节点组成，一个节点

代表一条语句或数条语句,节点间的连接称之为边,边表示语句间的控制流向。

3. 代码质量度量

软件质量的 ISO/IEC 9126 国际标准包括 6 个方面:功能性、可靠性、易用性、效率性、可维护性和可移植性。软件的质量是软件属性的各种标准度量的组合。对于软件开发人员来说,静态分析只是进行动态测试的预处理工作,并且静态分析已经成为一种自动化的代码校验方法。

4. 静态分析的任务

1) 发现程序的错误

程序的错误通常有以下一些。

(1) 使用了错误的局部变量或是全程变量。

(2) 不匹配的参数。

(3) 未定义的变量。

(4) 遗漏的标号或代码。

(5) 不适当的循环嵌套或分支嵌套。

(6) 不适当的处理顺序。

(7) 无终止的死循环。

(8) 不允许的递归。

(9) 调用并不存在的子程序。

(10) 不适当的连接。

2) 寻找潜伏问题的原因

潜伏问题的原因通常有以下一些。

(1) 未使用过的变量。

(2) 不会执行到的冗余代码。

(3) 可疑的计算。

(4) 潜在的死循环。

3) 提供程序的信息

程序的信息通常有以下一些。

(1) 每一类型语句出现的次数。

(2) 所用变量和常量的交叉引用表。

(3) 标识符的使用方式。

(4) 过程的调用层次。

(5) 程序代码违背编码规则。

4) 选择测试用例

软件测试的本质是对被测试的内容确定一组测试用例。测试用例包含如下信息。

(1) 输入有两种类型:前提(在测试用例执行之前已经存在的环境)和由某种测试方法所标识的实际输入。

（2）输出也有两类：预期输出和实际输出。

（3）测试活动要建立必要的前提条件，提供测试用例输入，观察输出，并且将实际输出与预期输出进行比较，以确定该测试是否通过。

（4）良好的测试用例信息（如图 2-1 所示）也支持测试管理。测试用例信息应该能够记录测试用例的执行历史，包括测试用例是什么时候由谁运行的，每次执行的通过/失败记录，测试用例所测试的是哪个软件版本。测试用例同样也需要被开发、评审、使用、管理和保存。

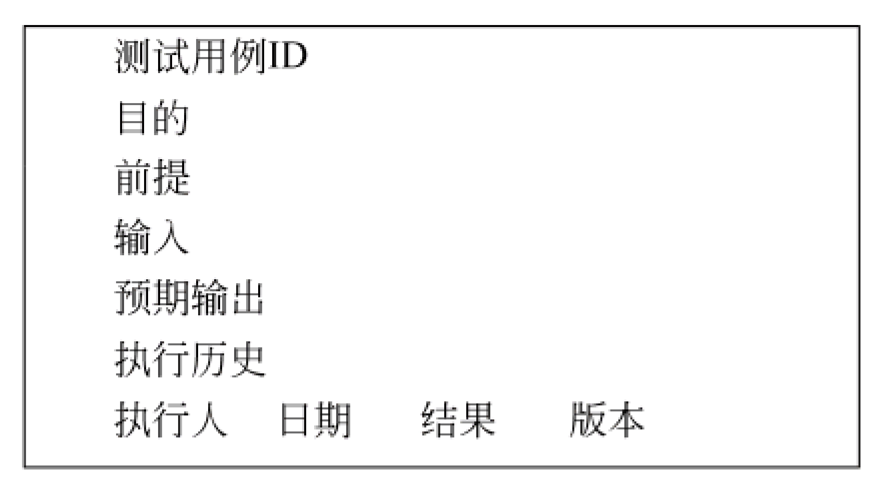


图 2-1　测试用例信息

5）为查错做准备

静态分析可为查错做准备，在静态分析过程中计算机并不运行被测试的程序，这是静态分析与人工测试的根本区别。

2.2　动态测试

1. 动态测试特点

动态测试是使被测代码在相对真实的环境下运行，从多个角度观察与检测程序运行时的功能、逻辑、行为和结构，并且通过实际运行的输出结果和预期输出结果的比较，来发现其中的错误。动态测试技术是借助工具进行测试的技术，主要特征是计算机必须真正运行被测试的程序，通过输入测试用例，对其运行情况（输入/输出的对应关系）进行分析。动态测试是在满足一定要求的样本测试数据上执行程序并分析实际输出以发现错误的过程，主要特点如下所述。

（1）运行被测试程序，获得程序运行的动态情况和真实结果，从而进行分析。

（2）必须生成测试用例来运行程序，测试质量与测试用例密切相关。

（3）生成测试用例、分析测试结果的工作量大，使得测试工作消耗较多。

（4）动态测试中涉及人员多、设备多、数据多，要求有较好的管理制度和工作流程。

2. 动态测试的内容

动态测试包括功能确认与接口测试、覆盖率分析、性能分析、内存分析等。

1）功能确认与接口测试

通过测试来确认各个单元功能的执行是否正确，其中包括单元接口、局部数据结构、重要的执行路径、异常处理的路径和边界影响条件等内容。

2）覆盖率分析

覆盖率分析主要是对代码的执行路径的覆盖范围进行评估。语句覆盖、判定覆盖、条件覆盖、条件/判定覆盖、修正条件/判定覆盖、基本路径覆盖都是从不同要求出发，为设计测试用例而提出依据。

3）性能分析

如果不能解决被测程序的性能分析问题，将降低并极大地影响被测程序的质量，于是查找和修改性能成为调整整个代码性能的瓶颈。性能分析需要利用工具，主要工具有软件的测试工具、硬件的测试工具（如逻辑分析仪和仿真器等）和软硬件结合的测试工具等。

4）内存分析

内存泄漏就是指申请了一块内存空间，使用完毕后没有释放掉。它的一般表现方式是程序运行时间越长，占用内存越多，最终占用全部内存，使整个系统崩溃。简单地说，由程序申请的一块内存，且没有任何一个指针指向它，那么这块内存就泄漏了。内存泄漏将导致系统运行的崩溃，尤其对于内存资源比较缺乏，应用非常广泛，而且往往又处于重要地位的系统，将可能导致无法预料的重大损失。通过测量内存使用情况，可以了解程序内存分配的情况，发现对内存的不正常使用，在系统崩溃前发现内存泄漏错误、内存分配错误，并精确显示发生错误时的前后情况，并指出发生错误的原由。

2.3 人工测试与机器测试

人工测试技术是指不依赖于具体的计算机，通过人工手段来进行测试的技术。而机器测试是指将大量的重复性工作由计算机完成，是一种依赖于计算机的软件测试技术，又称为机器测试技术。

据统计，使用人工测试方法能够有效地发现 30%～70%的逻辑设计错误和编码错误。由于人工测试技术在检查某些编码错误时，常常能够找出利用计算机不容易发现的错误，所以人工测试仍是一种有效的测试方法。人工测试的主要方法包括软件审查、代码检查和人工走查。

机器测试是基于计算机完成的测试技术，包括边界值测试、分支覆盖测试、原型和语法测试。其中语法测试是由语言编译器自动完成的。

2.3.1 软件审查

软件审查的对象是各开发阶段的成果，如需求分析、概要设计、详细设计等阶段的成果以及编码、测试计划和测试用例等。软件审查通常有以下几个步骤：制定计划、预审、准备、审查会、返工、终审。软件开发过程中的审查如图 2-2 所示。

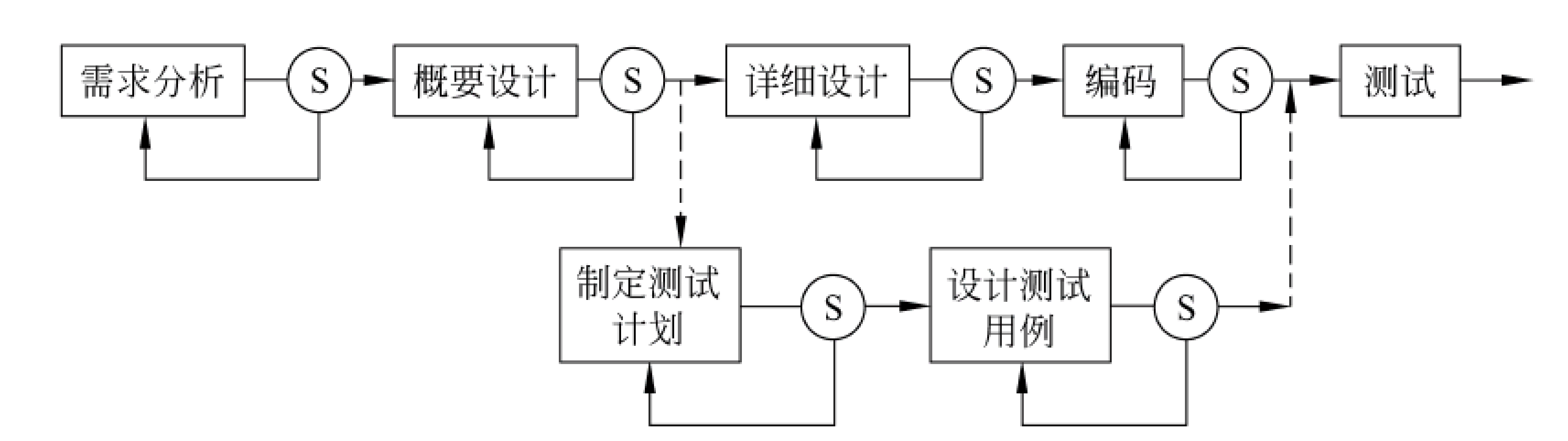


图 2-2　软件开发过程中的审查（用 S 表示）

1. 软件审查步骤

(1) 制定计划：在软件产品已具备阶段审查的条件后制定。首先要确定主持人，主持人负有组织审查会、最后决定审查结论的责任，主持人不应该是被审软件的开发人员。其次是确定参加审查的其他人员，软件审查组通常由一个主持人和4～5个其他人员（包括被评审对象的开发者）组成。

(2) 预审：是正审的初步，为了公正准确地完成评审，应该把与评审有关的资料提供给参审人员。

(3) 准备：开发人员收集有关的审查资料，并填写“软件审查概要”表。其他参审者应该阅读和研究开发人员所提供的资料，并填写“软件审查准备工作记录”。

(4) 审查会：首先由主持人了解会议准备情况，若认为准备工作不够充分，可以决定推迟会议召开时间；然后主持审查会，仔细记录所发现的不妥之处，由记录员将所发现的问题的位置、简要情况和问题类别等记录在“审查会发现问题报告”中；最后，参审人员对这次参审的结论进行评审，评审的结论可以是符合要求、需要返工和要求再次审查等。

(5) 返工：在审查会结束以后由开发者完成。把记录在“审查会发现问题报告”的错误改正过来。

(6) 终审：由主持人完成，检验所有需要改正的地方是否已经改正，并最后填写“审查结果报告”和“审查终结报告”。

2. 问题的表现形式

(1) 遗漏：在规格说明或标准中应该有的内容，但在送审资料中丢掉了。

(2) 多余：超出规格说明和标准，给出了多余的信息。

(3) 错误：内容有误的信息。

3. 审查中发现的问题

审查中发现的问题主要有以下一些。

(1) 接口问题。

(2) 数据问题。

(3) 逻辑问题。

(4) 输入和输出问题。

(5) 功能问题。

(6) 性能问题。

(7) 人为因素问题。

(8) 标准问题。

(9) 文档问题。

(10) 语法问题。

(11) 测试环境问题。

(12) 测试覆盖问题。

(13) 其他问题。

2.3.2 人工测试与机器测试的比较

人工测试不能保证测试的科学性与严密性，主要原因如下所述。

(1) 测试人员要负责大量文档、报表的制定和整理工作，工作烦琐。

(2) 受软件分发日期、开发成本及测试人员、资源等多方面因素的限制，难以进行全面的测试。

(3) 如果修正缺陷所花费的时间相当长，回归测试将变得更为困难。

(4) 对测试过程中发现的大量缺陷缺乏科学、有效的管理手段，责任含糊不清，没有人能向决策层提供精确的数据以度量当前的工作进度及工作效率。

(5) 反复测试带来的倦怠情绪及其他人为因素使得测试标准前后不一，测试花费的时间越长，测试的严格性也就越低。

(6) 难以对不可视对象或对象的不可视属性进行测试。

而机器测试不但可以满足测试的科学性，而且可以节约大量的时间、成本、人员和资源，并且测试脚本可以被重复利用。

2.4 黑盒测试

2.4.1 黑盒测试的概念

1. 黑盒测试的概念

黑盒测试是对软件的功能和界面的测试，其目的是发现软件需求或者设计规格说明中的错误，所以又称为功能测试，是一种基于用户观点出发的测试。在测试期间，把被测程序看做一个黑盒子，测试人员并不清楚被测程序的源代码或者该程序的具体结构，不需要对软件的结构有深层的了解，而是只知道该程序输入和输出之间的关系，依靠能够反映这一关系的功能规格说明书，来确定测试用例和推断测试结果的正确性。黑盒测试仅在程序接口处进行测试，只检查被测程序功能是否符合规格说明书的要求，程序是否能适当地接收输入数据并产生正确的输出信息。

黑盒测试可用于证实被测软件功能的正确性和可操作性。测试人员通过输入数据，然后观察输出的结果来了解被测软件的工作过程。通常测试员在进行测试时不仅使用可以输出正确结果的输入数据，而且还输入致使结果出错的输入数据，进而了解被测软件如何处理各种类型的数据。

黑盒测试有两种基本方法，即通过测试和失败测试，先进行通过测试，在进行通过测试时，实际上是确认软件能做什么，而不会去考验其能力如何。软件测试员只运用最简单、最直观的测试用例。失败测试或迫使出错测试是指采取各种手段来寻找软件缺陷，如为了破坏软件而设计和执行的测试用例。在失败测试进行之前，检测软件基本功能是否能够实现。在确信了软件的正确运行之后，就可以进行失败测试。

2. 黑盒测试要发现的问题

1）检测错误类型

黑盒测试仅考虑程序外部结构而不考虑程序内部逻辑结构，针对软件界面和软件功能进行测试。黑盒测试注重于测试软件的功能需求，主要检测下述几类错误。

（1）是否有不正确或遗漏了的功能。

（2）在接口上，输入能否正确地接收，并且能否输出正确的结果。

（3）是否有数据结构错误或外部信息（例如数据文件）访问错误。

（4）性能上是否能够满足最终需求。

（5）是否有初始化或终止性错误。

2）回答的问题

黑盒测试主要用于测试的后期，不考虑控制结构，主要回答下述问题。

（1）如何测试功能的有效性。

（2）何种类型的输入将产生好的测试用例。

（3）系统是否对特定的输入值敏感。

（4）如何分隔数据类的边界。

（5）系统能够承受何种数据率和数据量。

（6）特定类型的数据组将对系统产生何种影响。

3. 黑盒测试的主要内容

1）接收性测试

黑盒测试是从软件的接口接收测试输出结果，具有接收性测试的特点。

2）α/β测试

α测试是项目组内的成员对被测软件进行的测试，β测试是由项目组之外的人员参加的测试。α/β测试也适合于黑盒测试。也就是说，当测试发现错误后在开发人员修改的同时，项目经理也会对产品计划做出相应的调整，产品特征不断地修改。

3）发行测试

在正式发行前，产品要经过非常仔细的测试。除了专门的测试人员外，还需要几千个甚至几十万其他用户与合作者通过使用来对产品进行测试，然后将错误信息反馈到技术部门。到了发行测试时，如果出现非改不可的错误，就必须推迟软件的发行，在推迟时间内需要重新对软件产品进行全面的测试，将耗费大量的时间和人力物力。

4）回归测试

在此阶段，首先要检查以前找到的错误是否已经更正了。回归测试可使已更正的错误不再重现，并且不会产生新的错误。

5）RTM测试

RTM测试是指在产品发行阶段所进行的测试。在这一测试阶段，每一个错误都需要经过高端人员同意才能更正。因为这时候修改软件非常容易产生其他的错误，所以只有那种非修复不可的错误才将允许进行修改。如果在发行阶段软件还有许多严重的错误

的话，就不能按时发布。

4. 黑盒测试方法

黑盒测试方法主要有等价类划分、边界值分析、因果图、错误推测、状态测试等。等价类划分是指把所有可能的输入数据划分成若干个等价的子集，又称为等价类或等价区间，使得每个子集中的一个值在测试中的作用与这一子集中所有其他值的作用相同，使得在一个等价子集内任取一个测试用例就可以代表整个子集的测试效果，这样用少量的测试用例就可以达到用许多测试用例的测试效果，进而提高了测试效率。边界值分析不是从等价类中任取一个测试用例，而是从等价类子集中挑选处于边界的数据作为测试用例。因果图适合有多种输入条件的组合，相应产生多个动作的程序来设计的测试用例，因果图方法最终生成的是判定表，并依据判定表来设计测试用例。错误推测法是根据经验、直觉和预感来设计测试用例的，其效果与测试人员的经验有关。

在黑盒测试方法中，等价类划分和边界值分析是最常使用的测试方法，也是软件测试最基本的方法。

2.4.2　等价类划分

1. 问题的提出

在运行程序时虽然输入和输出结果不同，但是它们都通过了同样的软件的源代码路径。而通常一个源代码程序的路径是用于处理一定数值范围内的所有数值的，那么除了边界值以外，在边界值范围以内的所有数值在测试中对于检测软件缺陷的作用相同，因此可以把这一类的数看成是等价的，例如软件支持的字号范围是5～72。那么5和72之间的所有支持的字号都是等价类的测试用例。如果把所有可能的输入数据（有效的和无效的）划分成若干等价类，则在测试中每类中的一个典型值的作用与这一类中所有其他值的作用相同，因此，可以从每个等价类中只取一组数据作为测试数据，这样选取的测试数据最有代表性，而且最有可能发现程序逻辑中的错误，避免了测试数据的冗余。把所输入的条件划分成若干个等价类来对软件进行测试，那么这种对于输入条件集合的划分方法就是等价类划分方法。

2. 等价类划分概念

等价类划分以需求规格说明书为依据，不用考虑程序的内部结构，只参照对程序的要求和说明，通过分析说明书的各项需求，特别是功能需求，把对输入的要求和输出的要求区别开来，并把输入域分解成若干等价类，每类中一个典型值在测试中的作用与这一类中所有其他值的作用相同，因此可以在每个等价类中选用一组数据作为测试用例进行测试来发现程序中的错误。

等价类分为有效等价类和无效等价类两种类型。有效等价类是指对于程序的规格说明来说是合理的输入数据构成的集合。利用有效等价类可检验程序是否实现了规格说明中所规定的功能和性能。无效等价类与有效等价类的定义相反。设计测试用例时，要同时考虑这两种等价类。因为软件不仅要能接收合理的数据，也要能经受不合理数据的考

验,这样的测试才能确保软件具有更高的可靠性和坚固性。

等价类的划分是将程序的输入域划分为数据类,即把整个输入域集合划分成互不相交的一组子集,这些子集的并集就是整个输入域集合。这样在形式上既可以对输入域提供一种完备性,而互不相交性又可保证测试中对输入测试数据的无冗余性。如果输入域对象之间具有对称性、传递性或自反性,则它们等价,由它们组成的一个子集属于同一个等价类。由于同一个子集的元素具有等价关系,因此元素都有一些共同点,因此选取每个等价类中的一个元素就可以为一个测试用例。

3. 等价类划分方法的测试用例设计

使用等价类设计测试用例过程分为两个阶段:划分等价类阶段和选取测试用例阶段。

1) 划分等价类阶段

首先要把大量的输入数据划分成一系列的等价类,并注意区分有效等价类和无效等价类。等价类划分原则如下。

(1) 在输入条件规定了取值范围或数值的个数的情况下,则可以确立一个有效等价类和两个无效等价类。

例如,在规格说明中对输入条件规定:项数可以为 1～999,则有效等价类是 1≤项数≤999,两个无效等价类是项数<1 或项数>999。在数轴上的表示如图 2-3 所示。

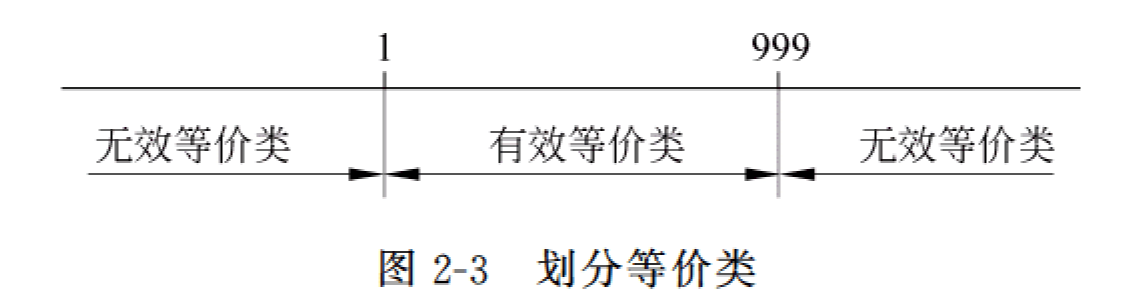


图 2-3　划分等价类

(2) 在输入条件规定了输入值的集合或者在规定的条件的情况下,可确立一个有效等价类和一个无效等价类。

例如,在 Java 语言中对变量标识符规定为“不能以数字开头,但可以包含字母、数字或下划线”。那么所有不以数字开头的构成有效等价类,而不在此集合内(以数字开头)的归于无效等价类。

(3) 在输入条件是一个布尔量的情况下,可确定一个有效等价类和一个无效等价类。

(4) 在规定了输入数据的一组值(假定 n 个),并且程序要对每一个输入值分别处理的情况下,可确立 n 个有效等价类和一个无效等价类。

例如,在教师上岗方案中规定对教授、副教授、讲师和助教分别计算分数,做相应的处理。因此可以确定 4 个有效等价类为教授、副教授、讲师和助教,一个无效等价类,它是所有不符合以上身份的人员的输入值的集合。

(5) 在规定了输入数据必须遵守的规则的情况下,可确立一个有效等价类和若干个无效等价类(不符合规则)。

又例如,Java 语言规定“一个语句必须以分号结束”。这时,可以确定一个“以分号结束”有效等价类,若干个无效等价类:“以冒号结束”、“以逗号结束”、“以空格结束”、“以

LF 结束”等。

(6) 在确知已划分的等价类中各元素在程序中处理方式不同的情况下，则应再将该等价类进一步划分为粒度更小的等价类。

【例 1】 考虑下面计算实数平方根的函数设计说明。

输入：实数。

输出：实数。

处理：当输入为 0 或大于 0 时，返回输入数的平方根；当输入小于 0 时，显示：“Square rooterror-illegal negative input”，并返回 0。

考虑平方根函数的测试用例区间，有 2 个输入区间和 2 个输出区间，表示如下：

输入分区：i<0，a≥0；

输出分区：ii≥0，b Error。

根据上述原则，可以把输入条件划分成两个等价类：n≥0 和 n < 0。从这两个等价类中可以选取两个输入数据作为测试用例。用这两个测试用例测试 4 个区间：

测试用例 1：输入 4，返回 2//测试区间 ii 和 a。

测试用例 2：输入－10，返回 0，输出“Square root error-illegal negative input”//测试区间 i 和 b。

上例的等价类划分较简单，但软件越复杂，等价类的确定就越难，等价类之间的相互依赖就越强，使用等价类划分设计测试用例技术难度越大。

2）选取测试用例阶段

首先建立等价类表，列出所有划分出的等价类，如表 2-1 所示。

表 2-1 等价类表

输入条件	有效等价类	无效等价类
……	……	……
……	……	……

然后从等价类中确定测试用例，选取测试用例可以按以下原则：设计一个测试用例覆盖尽可能多的未覆盖的有效等价类，以进一步减少测试次数；无效等价类必须每类一例，以防漏掉本来可能发现的错误。在等价分类中，代表一个类的测试数据可以在这个类的允许值范围内任意选择。

确定等价类划分测试用例的步骤如下。

S1：为每一个等价类规定一个唯一的编号，以便识别。

S2：设计一个新的测试用例，使其尽可能多地覆盖未覆盖的有效等价类；重复这一步骤，直到所有的有效等价类都被覆盖为止。

S3：设计一个新的测试用例，使其仅覆盖一个无效等价类，重复这一步骤，直到所有的无效等价类都被覆盖为止。

【例 2】 设一个程序的功能是读入代表三角形边长的 3 个整数，判断它们能否组成三角形，如果能够，则输出组成的三角形是等边、等腰或任意三角形的识别信息。

利用等价划分原则来划分等价类。表 2-2 列出了划分的结果，包括 8 个有效等价类和 5 个无效等价类。表 2-3 所列为测试用例。

表 2-2　等价类划分

输入条件	有效等价类	无效等价类
输入 3 个数	3 个数相等 3 个数中有两个数相等，a、b 相等 3 个数中有两个数相等，b、c 相等 3 个数中有两个数相等，c、a 相等 3 个数均不相等 两数之和不大于第三个数，其中最大的数为 a 两数之和不大于第三个数，其中最大的数为 b 两数之和不大于第三个数，其中最大的数为 c	1. 含有零的数据 2. 含有负整数 3. 少于 3 个整数 4. 含有非整数 5. 含有非数字符

表 2-3　测试用例

测试内容	测试数据	期望结果
等边	6,6,6	等边三角形
等腰	6,6,8;8,6,6;6,8,6	等腰三角形
任意	6,7,8	任意三角形
非三角形	3,4,8	不是三角形
零数据	0,8,6;8,6,0;8,0,6	
负数据	−6,7,8;6,−7, 8;6,7,−8	
遗漏数据	6,7,−	运行出错
非整数	6.6,7,8	
非数字符	A,7,8	

【例 3】 某城市的电话号码由三部分组成。这三部分的名称和内容分别如下。

地区码：空白或三位数字；

前缀：非‘0’或‘1’开头的三位数；

后缀：四位数字。

假定被调试的程序能接收一切符合上述规定的电话号码，拒绝所有不符合规定的号码，就可用等价分类法来设计它的调试用例，其过程如下。

S1：划分等价类，包括 4 个有效等价类，11 个无效等价类。表 2-4 列出了划分的结果。在每一等价类之后加有编号，以便识别。

表 2-4　电话号码程序的等价类划分

输入条件	有效等价类	无效等价类
地区码	空白(1)，3 位数字(2)	有非数字字符(5)，少于 3 位数字(6)，多于 3 位数字(7)
前缀	200～999 之间的 3 位数字(3)	有非数字字符(8)，起始位为“0”(9)，起始位为“1”(10)，少于 3 位数字(11)
后缀	4 位数字(4)	有非数字字符(12)，少于 4 位数字(13)，多于 4 位数字(14)

S2：确定调试用例。表 2-5 中有 4 个有效等价类，可以公用两个有效等价类。

表 2-5 有效等价类

调 试 数 据	范 围	期 望 结 果
( )276－2345	等价类(1),(3),(4)	有效
(635)805－9321	等价类(2),(3),(4)	有效

对于 10 个无效等价类，要选择 10 个调试用例，如表 2-6 所示。

表 2-6 无效等价类

调 试 数 据	范 围	期望结果	调 试 数 据	范 围	期望结果
(20A)123—4567	无效等价类(5)	无效	(777)145—6789	无效等价类(10)	无效
(33)234—5678	无效等价类(6)	无效	(777)34—6789	无效等价类(11)	无效
(7777)345—6789	无效等价类(7)	无效	(777)345—678A	无效等价类(12)	无效
(777)34A—6789	无效等价类(8)	无效	(777)345—678	无效等价类(13)	无效
(234)045—6789	无效等价类(9)	无效	(777)345—56789	无效等价类(14)	无效

4. 使用等价类划分设计测试用例

按照从等价类中选取用例的多少，可以把等价类测试分为弱等价类测试、强等价类测试、弱健壮等价类测试和强健壮等价类测试。

1）弱等价类测试

弱等价类测试使用一个测试用例中的一个变量。

2）强等价类测试

强等价类测试基于多个缺陷假设，因此需要等价类笛卡儿积的每个元素对应的测试用例。这些测试用例的模式与命题逻辑中的真值表构造具有相似性。笛卡儿积可保证两种意义上的完备性：一是覆盖所有的等价类；二是有可能地输入组合中的一个。

3）弱健壮等价类测试

健壮是指测试考虑了无效值的情况；弱是指考虑了有单缺陷假设的情况。

(1) 对于有效输入，使用每个有效类的一个值，就像在所谓弱一般等价类测试中所做的一样，这些测试用例中的所有输入都是有效的。

(2) 对于无效输入，测试用例将用一个无效值，并保持其余的值都是有效的。因此，“单缺陷”会造成测试用例失败。

4）强健壮等价类测试

健壮是考虑了无效值的情况，强是指考虑了多缺陷假设的情况。即从所有等价类笛卡儿积的每个元素中获取测试用例。在测试中，等价类测试的弱形式（弱等价类测试或弱健壮等价类测试）不如对应的强形式的测试全面。如果实现语言是强类型的，即无效值会引起运行时错误，则没有必要使用健壮形式的测试。如果错误条件非常重要，则适合进行健壮形式的测试。

2.4.3　边界值分析

1. 边界值分析的必要性

软件测试常用的一个方法是把测试工作按同样的形式划分。对数据进行软件测试，就是检查用户输入的信息、返回结果以及中间计算结果是否正确。实践表明，输入域的边界值比中间值更加容易发现错误。实践证明，大量的错误发生在输入或输出范围的边界上，而不是在输入范围的内部。因此针对各种边界情况设计测试用例，可以查出更多的错误。为此，边界值分析可作为一种测试技术。

2. 边界值分析

边界值分析也是一种黑盒测试方法，是一种补充等价划分的测试用例设计技术，它选择一组测试用例检查边界值。它不是选择等价类的任意元素，而是选择等价类边界的测试用例。在设计测试用例时，对边界处理必须给予足够的重视，为检验边界的处理而专门设计测试用例，常常可以取得良好的测试效果。提出边界条件时，一定要测试临近边界的合法数据，即测试刚好处于边界上的合法数据以及刚超过边界的非法数据。

边界值分析的基本思想是使用最小值、略高于最小值、正常值、略低于最大值和最大值作为输入变量值。

3. 利用边界值分析选择测试用例的原则

在用边界值来选择测试用例时，应按照下述原则。

(1) 如果输入条件规定了值的范围，则应该取刚达到这个范围的边界值，以及刚刚超过这个范围边界的值作为测试输入数据。

例如，重量在 10～50kg 范围内的邮件，可取 10 及 50，还可取 10.01、49.99、9.99、50.01 等作为测试用例。

(2) 如果输入条件规定了值的个数，则用最大个数、最小个数、比最大个数多 1 个、比最小个数少 1 个的数作为测试数据。

(3) 根据规格说明的每一个输出条件，使用规则(1)。

(4) 根据规格说明的每一个输出条件，使用规则(2)。

(5) 如果程序的规格说明给出的输入域或输出域是有序集合(如有序表、顺序文件等)，则应选取集合的第一个和最后一个元素作为测试用例。

(6) 如果程序用了一个内部数据结构，应该选取这个内部数据结构的边界上的值作为测试用例。

(7) 分析规格说明，找出其他可能的边界条件。

边界值分析关注输入空间的边界，并用以表示测试用例。其选择的基本原理是错误更可能出现在输入变量的极值附近。

4. 用边界值分析法举例

使用边界值分析方法设计测试用例，首先应确定边界情况。应当选取正好等于、刚刚大于或刚刚小于边界的值作为测试数据，而不是选取等价类中的典型值或任意值作为测试数据。

【例 4】 考虑下面计算实数平方根的函数的设计说明。

输入：实数。

输出：实数。

处理：当输入 0 或大于 0 时，返回输入数的平方根；当输入小于 0 时，显示："Square rooterror-illegal negative input"，并返回 0；库函数 Print_Line 用于显示出错信息。

考虑平方根函数，有 2 个输入区间和 2 个输出区间，表示如下：

输入分区：<0，a≥0；

输出分区：≥0，b Error。

用边界分析时大于等于 0 区间的边界是 0 和最大正实数，小于 0 区间的边界是 0 和最大负实数。输出区间的边界是 0 和最大正实数。根据边界值分析可以设计以下 5 个测试用例。

测试用例 1：输入最大负实数，返回 0，使用 Print_Line 输出"Square root error illegal negative input"//区间(i)的下边界。

测试用例 2：输入仅比 0 小的数，返回 0，使用 Print_Line 输出"Square root error illegal negative input"//区间(i)的上边界。

测试用例 3：输入 0，返回 0//区间(i)的上边界，区间(ii)的下边界和区间(a)的下边界。

测试用例 4：输入仅比 0 大的数，返回输入的正数的平方根//区间(ii)的下边界。

测试用例 5：最大正实数，返回输入的正数的平方根//区间(ii)的上边界和区间(a)的上边界。

对于复杂的软件，使用等价类划分不太实际，对于枚举型等非标量数据也不能使用等价类划分，如区间(b)并没有实际的边界。边界值分析还需了解输入数的底层表示，一种方法是使用任何高于或低于边界的小值和合适的正数和负数。

【例 5】 找零钱最佳组合。

假设商店货品价格(R)不大于 100 元(且为整数)，若顾客付款在 100 元内(P)，求找给顾客最少货币个(张)数？(货币面值有 50 元(N50)、10 元(N10)、5 元(N5)、1 元(N1)共 4 种)。

(1) 分析输入的情形如下。

$$R > 100$$
$$0 < R \leqslant 100$$
$$R \leqslant 0$$
$$P > 100$$
$$R \leqslant P \leqslant 100$$
$$P < R$$

(2) 分析输出的情形如下。

$$N50 = 1$$
$$N50 = 0$$
$$4 \geqslant N10 \geqslant 1$$

$$N10 = 0$$
$$N5 = 1$$
$$N5 = 0$$
$$4 \geqslant N1 \geqslant 1$$
$$N1 = 0$$

（3）分析规格中每一决策点的情形，以 RR1、RR2、RR3 表示计算要找 50、10、5 元货币数时的剩余金额。

$$RR1 \geqslant 50$$
$$RR2 \geqslant 10$$
$$RR3 \geqslant 5$$

（4）由上述的输入、输出条件组合出如下可能的情形。

$R > 100$

$R \leqslant 0$

$0 < R \leqslant 100$，　$P > 100$

$0 < R \leqslant 100$，　$P < R$

$0 < R \leqslant 100$，　$R \leqslant P \leqslant 100$，　$RR = 50$

$0 < R \leqslant 100$，　$R \leqslant P \leqslant 100$，　$RR = 49$

$0 < R \leqslant 100$，　$R \leqslant P \leqslant 100$，　$RR = 10$

$0 < R \leqslant 100$，　$R \leqslant P \leqslant 100$，　$RR = 9$

$0 < R \leqslant 100$，　$R \leqslant P \leqslant 100$，　$RR = 5$

$0 < R \leqslant 100$，　$R \leqslant P \leqslant 100$，　$RR = 4$

$0 < R \leqslant 100$，　$R \leqslant P \leqslant 100$，　$RR = 1$

$0 < R \leqslant 100$，　$R \leqslant P \leqslant 100$，　$RR = 0$

（5）为满足以上各种情形，测试用例设计如下。

① 货品价格＝101。
② 货品价格＝0。
③ 货品价格＝－1。
④ 货品价格＝100，付款金额＝101。
⑤ 货品价格＝100，付款金额＝99。
⑥ 货品价格＝50，付款金额＝100。
⑦ 货品价格＝51，付款金额＝100。
⑧ 货品价格＝90，付款金额＝100。
⑨ 货品价格＝91，付款金额＝100。
⑩ 货品价格＝95，付款金额＝100。
⑪ 货品价格＝96，付款金额＝100。
⑫ 货品价格＝99，付款金额＝100。
⑬ 货品价格＝100，付款金额＝100。

2.4.4 错误推测

1. 错误推测法的作用

使用边界分析法和等价划分技术，可以帮助开发人员设计具有代表性的、容易暴露程序错误的测试用例。但是，不同类型、不同特点的程序通常有一些特殊的容易出错的情况。此外，有时分别使用每组测试数据时程序都能正常工作，这些输入数据的组合却可能检测出程序的错误。一般说来，即使是一个比较小的程序，可能的输入组合数也往往十分巨大，因此必须依靠测试人员的经验和直觉，从各种可能的测试用例中选出一些最可能引起程序出错的方案。对于程序中可能存在哪类错误的推测，是挑选测试用例时的一个重要因素。

2. 如何使用错误推测法

错误推测法在很大程度上靠直觉和经验进行。它的基本想法是列举出程序中可能有的错误和容易发生错误的特殊情况，并且根据它们选择测试用例。对于程序中容易出错的情况也有一些经验总结出来，例如，输入数据为零或输出数据为零往往容易发生错误；如果输入或输出的数目允许变化(例如，被检索的或生成的表的项数)，则输入或输出的数目为 0 和 1 的情况(例如，表为空或只有一项)是容易出错的情况。还应该仔细分析程序规格说明书，注意找出其中遗漏或省略的部分，以便设计相应的测试用例，检测程序员对这些部分的处理是否正确。

错误推测法是用判定表或判定树把输入数据各种组合与对应的处理结果列出来进行测试；还可以把人工检查代码与计算机测试结合起来，特别是几个模块共享数据时应检查在一个模块中改变共享数据时，其他共享这些数据的模块是否能正确同步处理。

2.4.5 因果图

1. 问题的提出

等价类划分和边界值分析这两种方法并没有考虑到输入情况的各种组合，也没有考虑到各个输入情况之间的依赖关系。输入条件之间的相互组合，可能会产生一些新的情况。用前面两种测试方法时可以检测到各个输入条件可能出错的情况，但却忽略了多个条件组合起来时出错的情况。但要检查输入条件的组合不是一件容易的事情，即使把所有输入条件划分成等价类，它们之间的组合情况也相当多。因此必须考虑采用一种适合于描述对于多种条件的组合，相应产生多个动作的形式来考虑设计测试用例。这就需要利用因果图。

因果图考虑了多个输入之间的相互组合与相互制约关系，按一定步骤高效率地选择测试用例，同时还能指出程序规格说明中存在的问题。

2. 利用因果图生成测试用例的步骤

(1) 分析软件规格说明中，哪些是原因(即输入条件或输入条件的等价类)，哪些是结果(即输出条件)，并给每个原因和结果赋予一个标识符。

(2) 分析软件规格说明中的语义，找出原因与结果之间，原因与原因之间对应的关

系，根据这些关系画出因果图。

（3）由于语法或环境限制，有些原因与原因之间，原因与结果之间的组合情况不可能出现。为表明这些特殊情况，在因果图上用一些记号表明约束或限制条件。

（4）把因果图转换为判定表。

（5）以判定表的每一列为依据来设计测试用例。

3. 因果图中的基本符号

在因果图中，通常用 C_i 表示原因，它置于图的左部，表示输入状态；用 E_i 表示结果，它位于图的右部，表示输出状态；各节点表示状态，可取值 0 或 1，0 表示某状态不出现，1 表示某状态出现。图 2-4 所示的是因果图的图形符号。

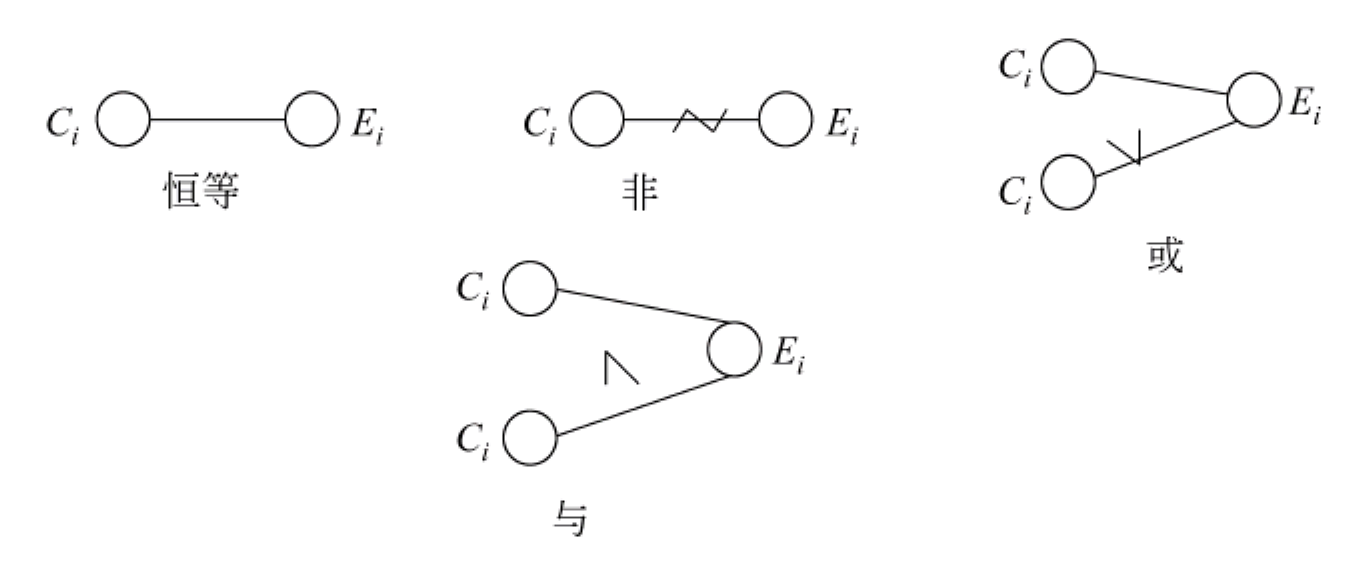


图 2-4　因果图的图形符号

（1）恒等：表示原因与结果之间的一对一的对应关系。若原因出现，则结果出现；若原因不出现，则结果不出现。例如：若 C_1 是 1，则 E_1 是 1；否则 E_1 为 0。

（2）非：表示原因与结果之间的一种否定关系。例如：若 C_1 是 1，则 E_1 是 0；否则 E_1 为 1。

（3）或：表示若几个原因中有一个出现，则结果出现；只有当几个原因都不出现时，结果才不会出现。例如：若 C_1 或 C_2 是 1，则 E_1 是 1；否则 E_1 为 0。“或”可以有任意个输入。

（4）与：表示若几个原因都出现，结果才出现；若几个原因中有一个不出现，结果就不出现。若 C_1 和 C_2 都是 1，则 E_1 为 1；否则 E_1 为 0。“与”也可以有任意个输入。

4. 输入的约束条件

在实际许多问题中，某些输入条件本身不可能同时出现，它们相互之间存在着某些依赖关系，称为约束。输出之间往往也存在约束。在因果图中有特定的约束符号表明这些约束，如图 2-5 所示。

（1）E(异)：表示两个原因不会同时成立，两个原因中最多有一个可能成立，即 a 和 b 不能同时为 1。

（2）I(包含)：表示 3 个原因中至少有一个必须为 1，即 a、b、c 不能同时为 0。

（3）O(唯一)：表示原因中有且仅有一个成立，即 a 和 b 有且仅有一个为 1。

（4）R(要求)：表示一个原因出现时，另一个原因也必须出现。不可能一个出现另一个不出现。即 a 出现时，b 必须出现；不可能 a 出现，b 不出现。

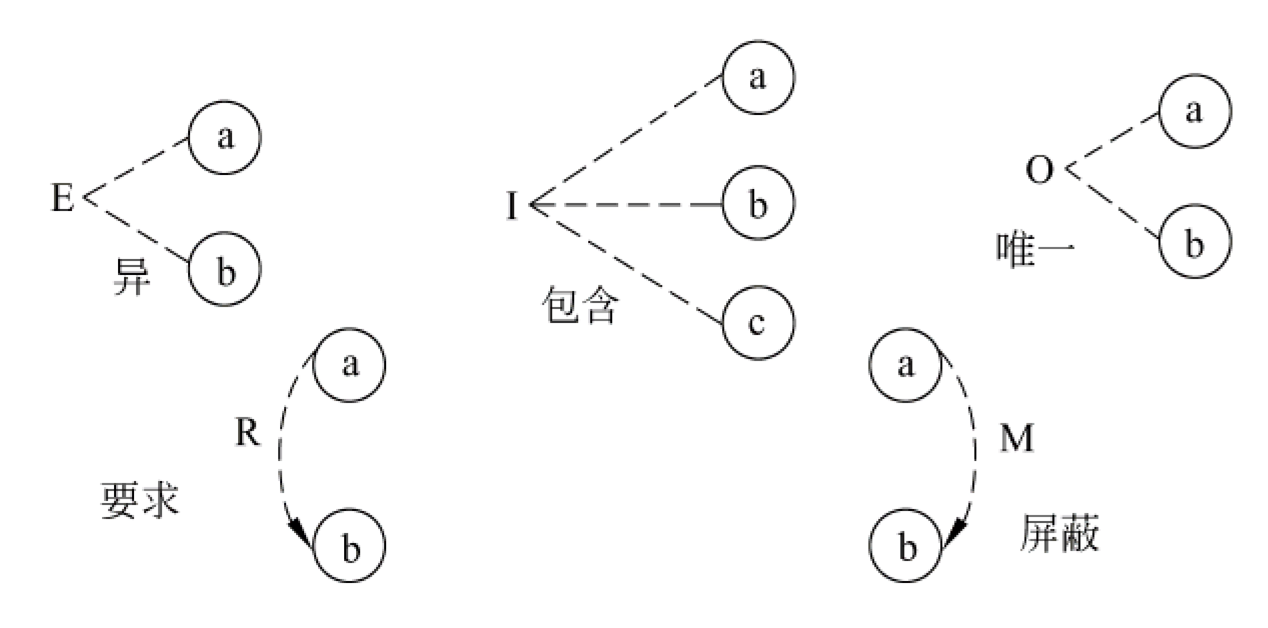


图 2-5 因果图的约束符号

(5) M(屏蔽)：表示一个结果成立时，另一个一定不成立，而当一个结果不成立时，另一个不一定。即当 a 是 1 时，b 必须是 0；而当 a 为 0 时，b 值不定。

5. 因果图类型划分

1) 形象因果图

对于常用的因果图，按其作用可称为特性因果图，按其形状可称为鱼刺图。因为它的突出特点是人们的形象逻辑思维，因此称它为形象因果图。

2) 数据因果图

随着测试领域的扩展，感到原来的因果之间定性的分析不够标准、不够具体和明朗，因此将大、中、小原因在图上尽量数字化表示，于是出现了数据因果图。数据因果图的优点是直观、可靠、可信，数据因果图确定要素准确，为制定策略提供充分依据。但要求作图者必须对现状、原因明确，为问题的解决提供比较充分的基础条件。目前来说数据因果图越来越受到人们的重视。

2.5 白盒测试

白盒测试要求测试人员清楚盒子内部的内容以及内部如何运作，也就是说白盒测试法是通过分析程序内部的逻辑与程序执行路线来设计测试用例的测试方法，因此白盒测试也被称为逻辑驱动测试，以测试的深度为主。由于这种方法按照程序内部的逻辑进行测试，检验程序中的每条通路是否都能按预定要求正确工作，所以白盒测试又称为结构测试。

白盒测试要求测试人员全面了解程序内部逻辑结构，以检查程序处理过程的细节为基础，对程序中尽可能多的逻辑路径进行测试，检验内部控制结构和数据结构是否有错、实际的运行状态与预期是否一致。在白盒测试中，测试人员必须检查程序的内部结构，从程序的逻辑入手，从而得出测试数据。白盒测试的主要方法有程序结构分析、逻辑覆盖、程序插装、域测试、符号测试和路径分析等。

2.5.1 白盒测试的作用

由于软件可能存在缺陷，所以要花费时间和精力来测试逻辑细节，软件存在的缺陷主

要包括以下方面。

(1) 逻辑错误和不正确的假设。当设计和实现主流之外的功能、条件或控制时，往往出现错误。

(2) 主观相信不可能执行某条逻辑路径，但在正常的情况下可能被执行。同时控制流和数据流的一些无意识的假设可能导致设计的错误，只有通过路径测试才能发现这些错误。

(3) 随机的错误。当一个程序被翻译成程序设计语言的源代码时，有可能产生某些错误，多数可被语法检查机制发现，但是还有些只有在进行白盒测试时才可被发现。

2.5.2　程序结构分析

如果使用白盒测试法对程序进行测试，必须首先了解这段程序的结构，才能保证后继的测试工作的进行。了解程序结构，必须进行结构分析，分析主要包括控制流分析和数据流分析两个方面。

1. 控制流分析

控制流分析是指用控制流图来表示程序控制结构。在程序流程图中，框内标明了处理要求或条件，而这些要求或条件在进行路径分析时并不是必要的。为了更加突出程序流程图的结构，需要简化程序流程图，于是出现了控制流图。在控制流图中只有两种符号：节点和控制流线。

(1) 节点：用标有编号的圆圈表示，表示程序流程图中的矩形框、菱形框和汇合点。

(2) 控制流线：以带箭头的弧线表示，它与程序流图中的流线一致，表明了程序控制的顺序。为了讨论和检查的方便，通常控制流线都标有名字。图 2-6 为一个简单的控制流图示例。

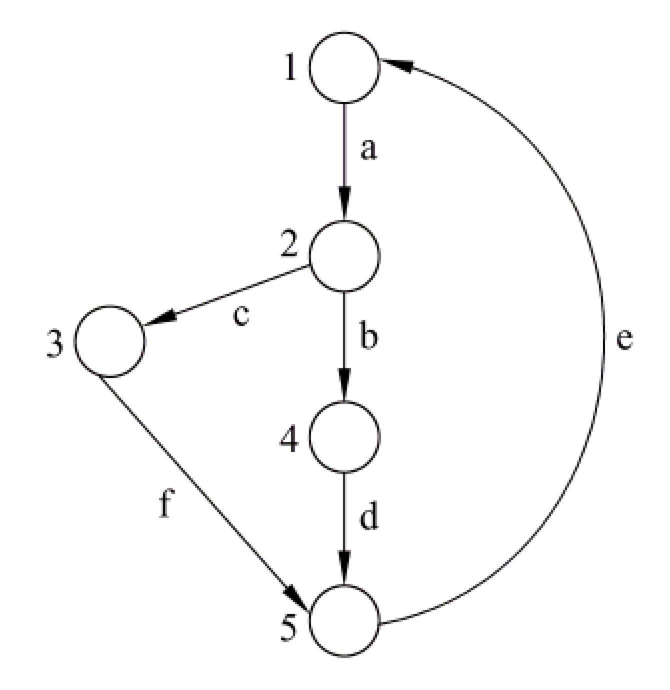


图 2-6　控制流图

在图中，1、2、3 等数字代表了控制流图中的节点，a、b、c 等代表了控制流线。图中一个节点包含一组语句，这组语句或者全部是顺序语句，或者是除最后一个语句是控制转移语句外其他语句都是顺序语句。一个节点也可以是程序中的一个汇合点。可以看出，通过分析程序的控制流图可以使测试人员非常清楚地知道待测程序的控制过程，从而为以后的测试工作打下坚实的基础。

2. 数据流分析

可以利用数据流分析、查找使用了未定义的变量错误，或定义的变量从未使用过等情况，这些都是程序错误的表现形式，如变量名混淆、拼错变量名字、丢失语句等。

如果程序中的某一语句执行时能改变程序中变量 V 的值，则称变量 V 被该语句定义。如果某一语句的执行引用了程序中变量 V 的值，则称该语句引用了变量 V。例如，在语句 X=Y+Z 中，定义了 X，引用了 Y 和 Z。

在控制流图中，每一语句要定义和引用的变量如图 2-7 所示，第一个节点定义了 3 个

变量 X、Y 和 Z。节点 2 使用了变量 W，而在此之前并未对其定义，这是一个错误。

节点	定义变量	引用变量
1	X,Y,Z	
2	Y	W,X
3	V	Y,Z
4	X	Z
5	Z	

图 2-7　定义和引用的变量

在节点 3 中定义了变量 V，但从未使用过，这表明存在一个异常。

从这个简单的例子中可以看出：通过对数据流的分析，测试人员就能比较容易地发现程序中存在的错误和异常。

2.5.3　逻辑覆盖

1. 逻辑覆盖分类

逻辑覆盖是指有选择地执行程序中某些代表性的通路，是对穷尽测试唯一可行的替代办法。逻辑覆盖是对一系列测试过程的总称，根据覆盖程序的详尽程度，大致分为下述几种覆盖。

(1) 语句覆盖：被测程序中的每一可执行语句在测试中尽可能都检验过，这是最弱的逻辑覆盖准则。图 2-8 所示的是实例程序中被测试模块的流程图。例如：

```
Begin
    If(A>1 and B=0)then x=x/4;
    If(A=2 or x>1)then x=x+1;
    Return;
End
```

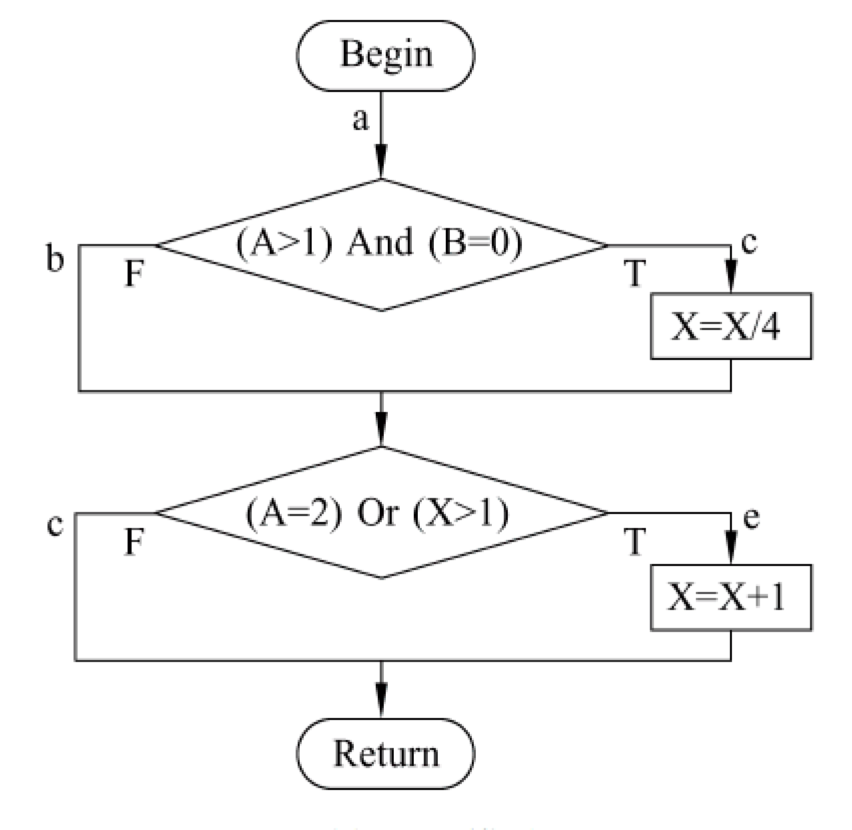


图 2-8　被测试模块流程图

由图中可以看出，为了使每个语句执行一次，程序执行的路径可以是：a→c→e；为此选择的用例为：A=2，B=0，X=3。语句覆盖对应的逻辑覆盖很少，在图 2-8 中两个判定条件都只测试了条件为真的情况，如果条件为假时处理有错误显然不能发现。此外，语句覆盖只关心判定表达式的值，而没有分别测试判定表达式中每个条件取不同值时的情况。测试每个语句只需两个判定表达式(A>1) AND (B=0)和(A=2) OR (X>1)都是真值，因此使用上述一组测试数据就够了。但是，如果程序中把第一个判定表达式中的逻辑运算符“AND”错写成“OR”，或把第二个判定表达式中的条件“>1”误写成“<1”，使用上面的测试数据并不能查出这些错误。语句覆盖是很弱的逻辑覆盖方法，为了更充分地测试，还需研究其他的测试方法。

(2) 判定覆盖：判定覆盖又称为分支覆盖，含义是不仅每个语句必须至少执行一次，而且每个判定的每种可能的结果都应该至少执行一次，也就是每个判定的分支都至少执行一次(真假分支均被满足一次)。对于图 2-8 可使用表 2-7 的测试数据。

表 2-7　表判定覆盖测试数据

数　　据	覆 盖 路 径	覆 盖 分 支	预期结果 X 值
A=2,B=0,X=4	a→c→e	TT	2
A=1,B=1,X=0	a→b→d	FF	0

判定覆盖比语句覆盖功能强，但是对程序逻辑的覆盖程度仍然不高，上面的测试数据只覆盖了程序全部路径的一半，但仍然可以看出判定覆盖比语句覆盖功能要强些。

(3) 条件覆盖：条件覆盖的含义是使判定表达式中的每个条件都取到各种可能的结果。

对于图 2-8 中的例子的条件，依次规定以下 8 种状态。

T1：A>1；F1：A≤1；T2：B=0；F2：B≠0；

T3：A=2；F3：A≠2；T4：X>1；F4：X≤1。

为了达到条件覆盖，测试选择如表 2-8 所示。

表 2-8　条件覆盖测试

数　　据	覆 盖 路 径	覆 盖 分 支	X 值
A=2,B=0,X=0	a→c→e	T1,T2,T3,F4	1
A=1,B=1,X=8	a→b→e	F1,F2,F3,T4	5

条件覆盖比判定覆盖功能强，因为它使判定表达式中的每个条件都取到了两个不同的结果，而判定覆盖却只关心整个判定表达式的值。但是，也可能有这样的情况：虽然每个都取到了两个不同的结果，判定表达式却始终只取一个值。例如，如果使用下面两组测试数据，则只满足条件覆盖标准并不满足判定覆盖标准，这是由于第一个判定都为假，第二个判定条件一个为真一个为假。

A=1,B=0,X=1 和 A=2,B=1,X=2。

(4) 判定/条件覆盖：既然判定覆盖不一定包含条件覆盖，条件覆盖也不一定包含判定覆盖，自然会有一种能同时满足这两种覆盖标准的逻辑覆盖，这就是判定/条件覆盖。它的含义是选取足够多的测试数据，使得判定表达式中的每个条件都取到各种可能的值，而且每个判定表达式也都可以取到各种可能的结果，具体见表 2-9。

表 2-9　判定/条件覆盖测试

数　　据	覆 盖 路 径	覆 盖 条 件	X 值
A=2,B=0,X=6	a→c→e	T1,T2,T3,T4	2.5
A=1,B=1,X=1	a→b→d	F1,F2,F3,F4	1

但是，这两组测试数据也就是为了满足条件覆盖标准最初选取的两组数据，因此，在某些情况下判定/条件覆盖也并不比条件覆盖更强。

(5) 条件组合覆盖：条件组合覆盖是更强的逻辑覆盖，它要求选取足够多的测试数据，使得每个判定表达式中的各种可能组合都至少出现一次。具体参见表 2-10。

表 2-10 条件组合覆盖测试

数　据	覆 盖 路 径	覆 盖 分 支
A=2,B=0,X=4	a→c→e	T1,T2,T3,T4
A=2,B=1,X=1	a→b→e	T1,F2,T3,F4
A=1,B=0,X=2	a→b→e	F1,T2,F3,F4
A=1,B=1,X=1	a→b→d	F1,F2,F3,F4

显然，满足条件组合覆盖标准的测试数据，也一定满足判定覆盖、条件覆盖和判定/条件覆盖。因此，条件组合覆盖是前述几种覆盖中最强的。但是，满足条件组合覆盖的测试数据并不一定能使程序中的每条路径都执行到。

(6) 路径覆盖：路径覆盖的含义是选取足够多的测试数据，使程序的每条可能路径都至少执行一次，如果程序控制流图中有环，则要求每个环至少经过一次。图 2-8 的程序控制流图如图 2-9 所示，具体的路径覆盖如表 2-11 所示。

表 2-11 路径覆盖测试

数　据	执 行 路 径
A=1,B=1,X=1	1,2,3
A=1,B=1,X=2	1,2,6,7
A=3,B=0,X=1	1,4,5,3
A=2,B=0,X=4	1,4,5,6,7

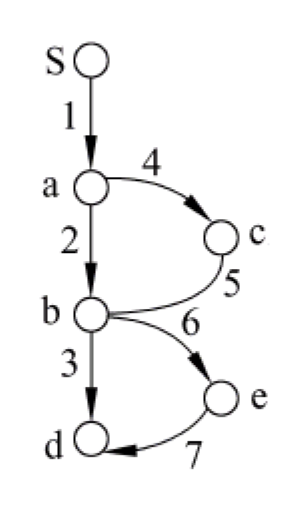


图 2-9 图 2-8 对应的控制流图

路径覆盖是相当强的逻辑覆盖，它保证了程序中每条可能的路径都至少执行一次，因此这样的测试数据更有代表性，发现错误的能力也比较强。但是，为了做到路径覆盖只需考虑每个判定表达式的取值，并没有检验表达式中条件的各种可能组合情况。如果把路径覆盖和条件组合覆盖结合起来，可以设计出检错能力更强的测试数据。对于图 2-8 的例子，只要把路径覆盖的第三组测试数据和前面给出的条件组合覆盖的四组测试数据联合起来，共有五组测试数据，就可以做到既满足路径覆盖又满足条件组合覆盖。

2. 覆盖准则

1) ESTCA 覆盖

覆盖就是要做得全面，没有遗漏。在下面所介绍的逻辑覆盖其出发点似乎合理。然而事实表明，覆盖并不能真正做到无遗漏。如果将：

```
…
If(i=0)
Then(i=j)
…
```

错写成：

```
...
If(i>0)
Then(i=j)
...
```

如果使用前面的覆盖，测试人员就无法发现这个问题。出现这种情况的原因在于，错误区域仅仅在 i=0 这个点上，即仅当 i=0 测试才能发现错误。这恰恰是在容易发生问题的条件判断那里未发现的错误。

针对这类情况，提出了 ESTCA(Error Sensitive Test Cases Analysis)覆盖，具体内容如下所述。

规则 1　对于 A rel B(rel 可以是<、=、>)型的分支谓词，应适当地选择 A 与 B 的值，使得测试执行到该分支语句时，A<B、A=B 和 A>B 的情况分别出现一次。

规则 2　对于 A rel_1 C(rel_1 可以是>或<，A 是变量，C 是常量)型的分支谓词，当 rel_1 为<时，应适当地选择 A 的值，使：

$$A=C-M$$

其中，M 是距 C 最小的机器允许的正数，若 A 和 C 均为整型，M=1。同样，当 rel_1 为>时，应适当地选择 A，使：

$$A=C+M$$

规则 3　对输入的变量赋值，使其在每一测试用例中均有不同的值与符号，并与同一组测试用例中其他变量的值与符号不一致。

显然，规则 1 是为了检测 rel 错误，规则 2 是为了检测“差一”这类的错误(如把 If A>1 错写成 If A>0)，规则 3 则是为了检测程序语句中的错误(如把引用的常量错写成引用的变量)。

在程序测试中，上述 3 条规则并不完备，但却非常有效。原因在于规则本身针对的是编程人员容易发生的错误或是错误频繁发生的区域，从而提高了发现错误的命中率。

ESTCA 规则的不足是在某些情况下不容易找到满足规则的输入数据，所以仍有很多错误发现不了。

2) LCSAJ 覆盖

LCSAJ(Linear Code Sequence and Jump Coverage)覆盖是一组覆盖率准则，是线性代码序列与跳转覆盖。

一个 LCSAJ 是一组顺序执行的代码，以控制流跳转为其结束点。LCSAJ 不同于判断—判断路径(DDP)。DDP 是根据程序有向图决定的。一个 DDP 是指两个判断之间的路径，但其中不能再有判断，程序的入口、出口和分支节点都可以是判断点。而 LCSAJ 的起点是根据程序本身决定的。它的起点既可以是程序的第一行、转移语句的入口点，也可以是控制流可以跳达的点。由此，几个 LCSAJ 首尾相接构成 LCSAJ 串，组成程序的一条路径。其中，第一个 LCSAJ 起点为程序起点，最后一个 LCSAJ 终点为程序终点。一条程序路径可以由两 3 个甚至多个 LCSAJ 组成。因此，根据 LCSAJ 与路径的这一关系，Woodward 提出了 LCSAJ 覆盖准则。

[第一层]：语句覆盖。

[第二层]：分支覆盖。

[第三层]：LCSAJ 覆盖。即程序中的每一个 LCSAJ 都至少在测试中经历过一次。

[第四层]：两两 LCSAJ 覆盖。即程序中每两个首尾相连的 LCSAJ 组合起来在测试中都要经历一次。

……

[第 n+2 层]：每 n 个首尾相连的 LCSAJ 组合在测试中都要经历一次。

显然，这个覆盖准则是分层的，而且越是高层的覆盖准则越难满足。

LCSAJ 覆盖要比判定覆盖复杂得多，而且维护 LCSAJ 的测试数据也是相当困难的，因为对一个模块的微小变动都可能对 LCSAJ 产生重大影响。

2.5.4 程序插装

程序插装是一种基本的测试手段，是借助向被测程序中插入操作(语句)来达到测试的目的的。那些被插入的语句称为“探测器”或“探针”。在程序特定部位插入“探针”的目的是为了把程序执行过程中发生的一些重要事件记录下来，如语句执行次数、变量值的变化情况、指针的改变等。

借助于程序插装方法，测试人员可以了解程序执行时的结构覆盖情况，如语句覆盖、判定覆盖、条件覆盖等其他一些覆盖的信息。在利用程序插装进行测试时需要考虑以下问题。

(1) 探测哪些信息。

(2) 在什么位置设置探测点。

(3) 需要设置多少个探测点。

对于前两个问题需要结合具体的程序解决，对于第三个问题，一般认为在没有分支的程序段中只需插入一个计数语句即可。但是由于待测程序一般都比较庞大，使用了多种控制语句，所以为了在程序中设计最少的计数语句，需要针对不同的控制结构进行具体的分析。下面列出了在测试中应在哪些部位设置计数语句。

(1) 程序块的第一个可执行语句之前。

(2) if、for、while 和 do…while、switch 语句的开始处。

(3) if、for、while 和 do…while、switch 语句结束之后。

(4) switch 语句中的每个 case 语句处。

(5) break、continue、return 和 exit 语句前。

2.5.5 符号测试

之所以普通的测试方法不容易查出程序中的错误，重要的原因就是测试点的选择比较困难。由于被测程序与规格说明可能存在差距，所以无论是用功能测试还是用结构测试方法都不能保证选取到完全有代表性的测试点。测试用例的选择成了模块测试的瓶颈，符号测试方法可以绕开这个问题进而达到问题的可解性。

符号测试的基本思想是允许程序的输入不仅可以是数值数据，也可以包括符号值。

符号可以是符号变量,也可以是包含这些符号变量的一个表达式。这样,在执行被测程序的过程中符号的计算就代替了普通测试执行中对测试用例的数值计算,所得的结果是符号公式或符号谓词。

如果原来测试某程序时,要从输入数据 X 的取值范围 1～200 中选取一个进行数值计算,现在可以用 X1 作为输入数据,代入程序进行代数运算。这样所得的结果是含有 X1 的代数表达式,对于判断程序的正确性就更为直观。同时,进行一次符号测试等价于选用具体数值数据进行了大量普通测试。例如,上述对 X1 的测试就等价于进行了 200 次普通的测试。

符号测试是程序测试与程序验证的一个折中方法。一方面,它沿用了传统的程序测试方法,通过运行被测试程序来检验它的可靠性。另一方面,由于一次符号测试的结果代表了一大类普通测试的结果,证明了程序接收此类输入后,所得的输出结果是否正确。理想的情况是程序中仅有有限的几条可执行路径,若对这几条路径都完成了符号测试,确认程序正确性的可能就更大。

符号测试方法的优点是:可以很容易地确定所给的一组测试用例是否覆盖了程序的各条路径。对于任何一组测试用例,可以首先确定它所经历的测试路径,然后,再给出输入变量的符号值,进而得到路径条件。

2.5.6 程序变异

事实上,几乎不可能找出程序中所有的错误,现实的解决办法是将搜索错误的范围尽可能地缩小,以利于测试某类错误是否存在。基于这一思想,出现了错误驱动测试方法。错误驱动测试方法可以把目标集中在对软件危害最大的可能错误,而暂时忽略对软件危害较小的可能错误。这样可以取得较高的测试效率,并能降低测试的成本。

程序变异方法是一种错误驱动测试,错误驱动测试分为程序强变异和程序弱变异。

1. 程序强变异

程序强变异又称为程序变异,由 R. A. DeMillo 和 T. A. Budd 等人最早提出。Demillo 认为,当程序被开发并经过简单测试后,残留在程序中的错误不再是很严重的错误,而是一些难以发现的小错误,例如遗漏了某个操作、分支谓词规定的边界有位移、运行时的内存错误等。即使是一些较复杂的错误,也可以是由这些简单错误组合而成的。程序变异的目标就是查出这些简单的错误及其组合错误。

程序变异因子的定义如下。

程序 P 的变异因子 m(P)也是一个程序,它是对 P 进行微小改动而得到的。因而,也可以说 m(P)是 P 的一个变换。

如果程序 P 是正确的,则 m(P)是一个几乎正确的程序;如果 P 不正确,则 P 的某一个变异因子 m(P)也可能是正确的。

程序变异的基本思想是:假设 P 有一组测试数据,其集合为 D。若 P 在 D 上是正确的,则可以找出 P 的变异因子的某一集合:

$$m=\{M(P)\mid_{M(P)}\text{是 P 的变异因子}\}$$

若 m 中每一元素在 D 上都存在错误，则可以认为程序的正确程度较高。若 m 中的某些元素在 D 上存在错误，则可能存在以下 3 种情况。

(1) 变异因子与 P 在功能上等价。

(2) 现有的测试数据不能找出变异因子间的差别。

(3) 虽然 P 可能含有错误，而某些变异因子却可能是正确的。

测试人员应该核对程序及其变异因子，尽力避免第一种情况出现。第二种情况告诉测试人员，当 P 与某些变异因子都正确时，可能是因为测试数据太少，并且不够典型造成的。此时，应该增加测试数据，直到所有的变异因子都出错。第三种情况提醒测试人员，当许多典型的测试数据仍然不能使某一变异因子出错时，此变异因子就有可能是程序的正确形式。

使用程序变异方法，表明对测试数据集合里的每一个元素，都要对程序 P 及其变异因子进行测试。因此，必须对测试数据集合 D 和变异因子集合 m(P)进行精心挑选，这也是强变异方法成功的关键。所以变异因子是变异操作符作用在被测试程序上的结果。变异操作符接收被测程序的输入，而产生一系列不同的变异因子。

如果把程序处理的数据元素看成是常量、标量或数组，那么可在数据元素被引用时用其他的数据元素替换。如可以将常量增加一点，或是减少一点，也可以将数组变量名换成另一相同维数的数组变量名，还可以将操作符作些变换、替换或删除语句，改变循环条件等。

对程序进行变换的方式是多种多样的，而且变换依赖于被测程序使用的设计语言。对程序变换的选择与测试人员的实践经验有关。通过变异分析构造测试数据的过程是一个循环进行的过程，测试人员首先提供被测程序以及初始数据，其次还要提供用于程序的变异运算符。当由此产生的变异因子和程序本身被初始测试数据测试后，可能有某些变异因子未能发现错误。这时，用户需要增加测试数据。

一些常用的变异运算如下。

(1) 常量之间的替换。

(2) 数组之间的替换。

(3) 将常量替换为数组分量。

(4) 将数组分量替换成常量。

(5) 同维数的数组名之间的替换。

(6) 算术运算符之间的替换。

(7) 关系运算符之间的替换。

(8) 逻辑运算符之间的替换。

(9) 插入绝对值符号。

(10) 插入单目运算符。

(11) 语句分解。

(12) 语句删除。

(13) 循环终止条件变换。

程序变异方法也有以下两个缺点。

(1) 要运行所有的变异因子,因而提高了测试成本。

(2) 决定程序与其变异因子是否等价是一个递归不可解的问题。

2. 程序弱变异

程序弱变异的目标是要查出某一类错误,其主要思想如下。

设 P 是一个程序,C 是程序 P 的简单组成部分。若有某一变换作用于 C 而生成 C′,而 P′是含有 C′的 P 的变异因子,则在弱变异方法中,使用的测试数据必须满足要求:当 P 在此测试数据下运行时,C 被执行,且至少在一次执行中,使 C 产生的值与 C′不同。

从以上的叙述中可以看出,弱变异和强变异的主要差别在于:弱变异强调变动程序的组成部分,根据弱变异的准则,只要事先确定导致 C 与 C′产生不同值的测试数据集合,则可将程序在此测试数据集合上运行,并不实际产生其变异因子。

在弱变异方法的实现中,关键问题是确定程序 P 的组成部分以及与其有关的变换。组成部分可以是程序中的计算结构、变量定义与引用、算术表达式、关系表达式以及布尔表达式等。结合实际测试可归结 5 种最基本的程序组成部分:变量引用、变量定义、算术表达式、关系表达式和布尔表达式。不难看出:前两部分可以包含在后三部分中,算术表达式可包含在关系表达式中,而关系表达式又可包含在布尔表达式中。

1) 变量引用

变量引用包括变量值的使用,变量引用的弱变异使得被引用的变量变为另一变量。如在语句 X=Y 中,引用的 Y 变为了另一程序的变量。假设 V 是程序 P 中变量引用组成部分 C 的一个被引用变量,C′是 C 的一个变换。要保证 C 和 C′在测试数据上执行时产生的值不同,必须要求程序执行到 C 时,所有程序变量的值均与 V 的值不相同。

因此,要保证在一变量引用中,被引用的变量是正确变量,就必须产生测试数据集合,使得运行到此变量时,被引用变量不与任何其他程序变量的值相等。但要让所有的程序变量都不与被引用变量相等,这就需要很多组测试数据。实际上,由于程序中存在很多变量引用组成部分,要对所有组成部分进行弱变异测试,是非常耗费时间的。较为经济可行的办法是,将变量引用组成部分局限在错误数组元素引用与错误输入变量引用两种元素上。对两种元素进行弱变异的准则是:选取测试数据,使程序运行到该组成部分时,数组的各元素均不相等,或是程序的形参互不相等。

2) 变量定义

变量定义指的是给变量赋以新值。如在语句 X=Y 中,变量 X 被赋予了新值 Y。这条语句被称为变量定义的组成部分。

3) 算术表达式

算术表达式的弱变异需要考虑三类常见错误:表达式与正确表达式相差一个常数;表达式是正确表达式的常数倍;表达式的系数有错误。显然,后一种情况包括了前两种情况。但前两种情况要求较少的测试数据,因而将其单独分类。

4) 关系表达式

关系表达式的弱变异只需考虑两类简单的错误:关系运算符错误和相差一个常数的错误。

5）布尔表达式

布尔关系是用 NOT、AND 和 OR 这 3 个逻辑运算符连接起来的复杂关系。设 B 是一个布尔关系：

$$B = L(E_1, E_2, \cdots, E_n)$$

其中，$E_i(i=1,2,\cdots,n)$是算术关系，L 是含有 NOT、AND 或 OR 运算符的逻辑表达式，B 的一个变换为：

$$B' = L'(E_1, E_2, \cdots, E_n)$$

要使 B 与 B' 区别开来，使用的测试值必须使得$(E_1, E_2, \cdots, E_n)$是各种不同的真值组合。

弱变异测试相对于强变异测试，可以看作是一种测试数据选择准则。其优点是：用一组测试数据就可以检验出多个程序组成部分的正确性，从而可以减少测试中程序运行的次数；另外，弱变异的准则对测试数据的选择有一定的指导作用，用户可以有针对性地选择测试数据。其缺点是：当某一组成部分 C_1 有错，而另一部分 C_2 也有错时，程序在能检验出 C_1 错误的测试数据 T 下运行时，可能会由于 C_2 的作用，使得程序运行结果正确。但强变异测试会要求删除所有的变异因子，所以不会出现这种情况。

2.6 白盒测试和黑盒测试的比较

白盒测试考虑了黑盒测试不考虑的方面。同样地，黑盒测试也考虑了白盒测试不考虑的方面。白盒测试只考虑测试软件产品，它不保证完整的需求规格是否被满足。而黑盒测试只考虑测试需求规格，它不保证实现的所有部分是否被测试到。黑盒测试会发现遗漏的缺陷，指出规格的哪些部分没有被完成。而白盒测试会发现逻辑方面的缺陷，指出哪些实现部分是错误的。图 2-10 所示的文氏图表明了黑盒测试与白盒测试能够发现的错误。其中：

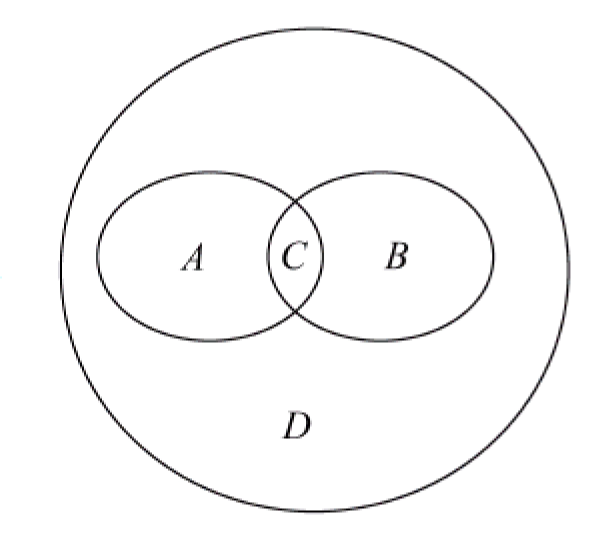


图 2-10 黑盒测试与白盒测试能够发现的错误

A——代表只能用黑盒测试发现的错误；

B——代表只能用白盒测试发现的错误；

C——代表用黑盒测试和白盒测试都能发现的错误；

D——代表黑盒测试和白盒测试均无法发现的错误；

$A+C$——代表能用黑盒测试发现的错误；

$B+C$——代表能用白盒测试发现的错误；

$A+B+C$——代表用黑盒测试和白盒测试能发现的错误；

$A+B+C+D$——代表软件中的全部错误。

白盒测试比黑盒测试成本高。白盒测试需要在测试计划前产生源代码，并且在确定合适的数据和软件是否正确方面需要花费更多的工作量。尽可能使用可获得的规格从黑盒测试方法开始测试计划。白盒测试计划应当在黑盒测试计划成功通过之前就开始，使用已经产生的流程图和路径判定。路径应当根据黑盒测试计划进行检查并且决定和使用额外需要的测试。

2.6.1 白盒测试的特点

1. 白盒测试的优点

(1) 能仔细考虑软件的实现。

(2) 可以检测代码中的每条分支和路径。

(3) 揭示隐藏在代码中的错误。

(4) 对代码的测试比较彻底。

2. 白盒测试的缺点

(1) 昂贵。

(2) 无法检测代码中遗漏的路径和数据敏感性错误。

(3) 不验证规格的正确性。

2.6.2 黑盒测试的特点

1. 黑盒测试的优点

(1) 对于子系统甚至系统,效率要比白盒测试高。

(2) 测试人员不需要了解实现的细节,包括特定的编程语言。

(3) 测试人员和编程人员彼此独立。

(4) 从用户的视角进行测试,很容易理解和接受。

(5) 有助于暴露规格的不一致或有歧义的问题。

(6) 测试用例可以在规格完成之后马上进行。

2. 黑盒测试的缺点

(1) 只有一小部分输入被测试到,要测试每个可能的输入几乎不可能。

(2) 没有清晰、简明的规格,测试用例很难设计。

(3) 如果测试人员不被告知开发人员已经执行过的用例,在测试数据上会存在不必要的重复。

(4) 有很多程序路径没有被测试到。

(5) 不能直接针对特定程序段测试,而这些程序段可能很复杂,有可能隐藏更多的问题。

(6) 大部分和研究相关的测试都是直接针对白盒测试的。

白盒和黑盒测试方法是基于完全不同的观点,反映了事物的两个极端。两种方法各有侧重和优势,但不能彼此替代。在现代的测试理念中,这两种测试方法不是截然分开的,而是交叉使用的。灰盒测试就是介于白盒测试和黑盒测试之间的测试,最常见的灰盒测试是集成测试。

2.7 敏捷测试方法简介

随着敏捷开发的出现与应用,软件测试也在发生着巨大的变化。传统的软件测试已不能适应当前的开发方式,需要新的理论和方法来推进软件工程的进步。

2.7.1 敏捷技术概述

在传统的软件开发方法中,认为需求在项目初期分析清楚并且保持稳定不变,这种想法不能快速地将需求变化融合到软件开发中,意味着对业务环境反应迟钝,最终将导致失败。因此,为了寻找软件程序能够随时适应需求而调整的方法,敏捷开发技术应运而生了。

在 2001 年,以 Kent Beck、Martin Fowler、Robert C. Martin 及 Ward Cunningham 等为首的软件工程专家成立了敏捷联盟(Agile Alliance),并提出了如下所述的敏捷过程的理念。

(1) 人和交互重于过程和工具。

(2) 可以工作的软件重于求全责备的文档。

(3) 客户合作重于合同谈判。

(4) 随时应对变化重于循规蹈矩。

1. 敏捷技术的特点

(1) 敏捷型方法具有适应性而非预见性。工程方法试图对一个软件开发项目在很长的时间跨度内做出详细的计划,然后依计划进行开发。这类方法在一般的情况下工作良好,但当需求、环境等有变化时就不能保证。因此其本质上拒绝变化。而敏捷型方法则不拒绝变化。其目的就是能够适应变化的过程,并通过自身的改变来适应变化。

(2) 敏捷型方法是面向人的,而面向过程的工程型方法的目标是定义一个过程。而敏捷型方法认为没有任何过程能代替开发组的技能,过程起的作用是对开发组的工作提供支持。

敏捷开发过程指的就是一种与传统的瀑布模型开发和软件开发的能力成熟度模型(Capability Maturity Model,CMM)所追求的严谨的文档制度截然相反的开发过程。这一开发过程注重开发团队和成员之间的关系而不是以开发的进程和使用的工具为重点,注重所开发的软件产品而不是追求广泛的文档编制,注重开发过程中与客户的协同工作而不是以签订合同的谈判为工作的核心,注重在开发过程中随时调整计划而不是同意完全遵循某一开发计划,以实现开发过程的敏捷。

2. 敏捷测试现状

敏捷方法的发展,打破了传统的瀑布模型开发,改变了整个软件开发过程中的角色和定位。由于在敏捷开发的初期,主要依靠开发人员来进行推动。很多测试人员不了解敏捷方法,仍然习惯了按照传统的瀑布模式进行软件测试,即按照 V 模型所指导的步骤进行测试,保证软件与需求、设计的相符合,但这样很容易形成一种测试思维的定式。当用户需求不明确、需求变化较快时,沿用传统测试方法的测试人员将变得无所适从。

目前比较流行的敏捷测试方法有测试驱动开发和相关环境驱动测试等。还有很多按照敏捷的原理为软件测试开发的相应的测试框架,其中有 Kent Beck 等提出的 xUnit 系列单元测试框架和 Ward Cunningham 等提出的 Framework for Integrated Test(FIT)集成测试框架。

3. 敏捷测试的理念

（1）敏捷测试说明了一种测试思想。当它应用于敏捷测试的环境时，能够强化开发的质量。敏捷测试需要多种成熟理论的支持，如建立测试模型或是画出测试流图等。

（2）敏捷测试是对已有方法的补充，而不是一个完整的方法论。

（3）敏捷测试是一种有效的共同工作的方法。

（4）敏捷测试可以有效地使用受限资源。

（5）敏捷测试强化单元测试理论和实现方式。

（6）敏捷测试是改进众多软件开发成果的有效技术。

（7）敏捷测试是面向开发人员和测试人员的。

（8）敏捷测试产生的输出工件（测试计划、测试用例、测试数据等）会在测试代码中得到具体的体现，而无须生成其他大量的、难以管理的文档。

（9）敏捷测试工具能够帮助测试人员提高效果，可来自于供应商，也可能测试者自己书写。

（10）敏捷测试并不适合每一个人。

2.7.2 敏捷测试的原则

敏捷测试的原则为软件开发过程中的测试奠定了基石，主要内容如下所述。

（1）敏捷在于角色，将测试中参与的角色和职责转化为推动敏捷测试的原动力。

（2）有目的的测试，要了解测试的对象是什么。

（3）细化测试计划。细分成了两种格式，一种是文档格式；另一种是具体时间进度格式。

（4）用例的精化。

（5）多种测试方法组合。

（6）了解测试工具。

（7）引导资源投入。

（8）可持续进行过程。

（9）主张复用。

（10）紧密追踪。

（11）尽早发现更多的缺陷是主要目标。

（12）快速反馈。

（13）快速更新。

（14）高质量的工作。

（15）过程比结果更重要。

（16）局部调整。

（17）开放诚实的沟通。

（18）利用人的直觉。

2.7.3 敏捷测试的意义

敏捷测试并不适合于所有情况。即使是条件非常适合,也不能保证它能良好地运行,敏捷测试需要满足以下一些条件才可能发挥其效力。

(1) 敏捷测试不是一门完整的方法论,它只是某个软件开发过程的一小部分。

(2) 采用迭代式、递增式的开发方式。

(3) 严重不足的资源量投入,最佳的选择就是采用敏捷方法来测试软件。

(4) 完全介入单元测试中,这是敏捷测试的核心之一。

(5) 成熟的培训体制。敏捷测试更依赖于一套完整的培训体制,敏捷测试团队部分人员并不是专职的测试者,所以当非职业测试者进行角色转化时,需要进行大量测试意识和方法培训。

(6) 有真正的敏捷测试意识。不论何时,接受新的事物。

(7) 需要负责、主动的人员。敏捷测试团队形成了统一的目标后就会高强度地聚合在一起。这就要求团队人员能够相互协同、相互信任、相互帮助进行高质量的工作。

(8) 良好的沟通环境。敏捷测试需要团队间的紧密合作,测试人员的工作环境是开放式的,有利于沟通。

小　　结

软件测试的目的是为了找到软件的错误和缺陷,首先要根据软件开发各个阶段的规格说明和程序的内部结构设计测试用例集合,用测试用例集执行程序,以发现程序的错误。软件测试是通过系统的测试方法,发现软件中的错误,提供丰富的错误诊断信息,以便于改正错误,进而预防错误的发生,减少软件开发的费用。软件测试的基本方法有静态分析、动态测试、人工测试、自动测试、黑盒测试和白盒测试等。黑盒测试和白盒测试两种方法都是根据一组测试用例作为输入去执行程序,对程序的行为进行检验,确定其是否与预期的结果一致。

黑盒测试不考虑程序的内部逻辑结构和内部特性,只依据程序的需求规格说明书,检查程序的功能是否达到和符合它的功能说明,是否与功能要求完整一致,又称为功能测试方法。在设计测试用例时,只考虑软件的功能要求,不涉及程序的内部结构和实现细则,而且该测试用例应该能检验程序的全部功能。黑盒测试可以发现不正确的或漏掉的功能、接口错误、数据结构或外部数据库访问中的错误、性能错误、初始化错误和终止错误。

白盒测试是利用程序内部的逻辑结构及有关的信息,设计测试用例,对程序的所有逻辑路径进行测试。白盒测试又称为结构测试方法,它是从程序结构出发来设计测试用例的。白盒测试的测试用例用于检查模块中的独立路径,检查每个逻辑判断的情况;检查每个循环变量的初值、中间值和终值,检查程序的内部数据结构是否有效。通过白盒测试可以发现程序的逻辑错误和不正确的假设和条件、没有预料到的路径、语法检查未发现的错误。

通过静态分析、动态测试、人工测试、黑盒测试、白盒测试等主流技术的学习，可以为掌握软件测试方法建立坚实的基础。

习　题　2

1．在白盒技术的测试用例中，(　　)是最弱的覆盖标准。
　A．语句覆盖　　B．路径覆盖　　C．条件组合覆盖　　D．判定覆盖

2．设被测试的程序段为：

```
{
    S1;
    If((X=0&&(Y=2))S2;
    Else if((X<1)||(Y=1))S3;
    S4;
}
```

找出实现下列测试方法至少要选择的数据组。
　(1) 语句覆盖　　(2) 条件覆盖　　(3) 判定覆盖

可供选择的测试数据组

组	X	Y
1组	0	2
2组	−1	3
3组	3	2
4组	2	1

3．黑盒测试法和白盒测试法用于软件测试阶段，其中黑盒测试主要用于测试软件的(　　)。
　A．结构合理性　　B．程序正确性
　C．程序外部功能　　D．程序内部逻辑

4．白盒测试技术的方法有________、________、________、________、________和________。

5．说明敏捷测试的基本过程。

第 3 章　软件测试过程

学习要点：

❖ 单元测试。
❖ 集成测试。
❖ 回归测试。
❖ 确认测试。
❖ α 测试与 β 测试。
❖ 系统测试。

软件测试过程分成 4 个步骤：单元测试、集成测试、确认测试和系统测试，每一步是前一步的继续，其顺序关系如图 3-1 所示。在软件系统底层进行的测试为单元测试或模块测试，经过单元测试，底层软件缺陷被找出并修复之后，就组成在一起，对模块的组合进行集成测试，这个不断增加的测试过程继续进行，加入越来越多的软件片段，直至整个软件。

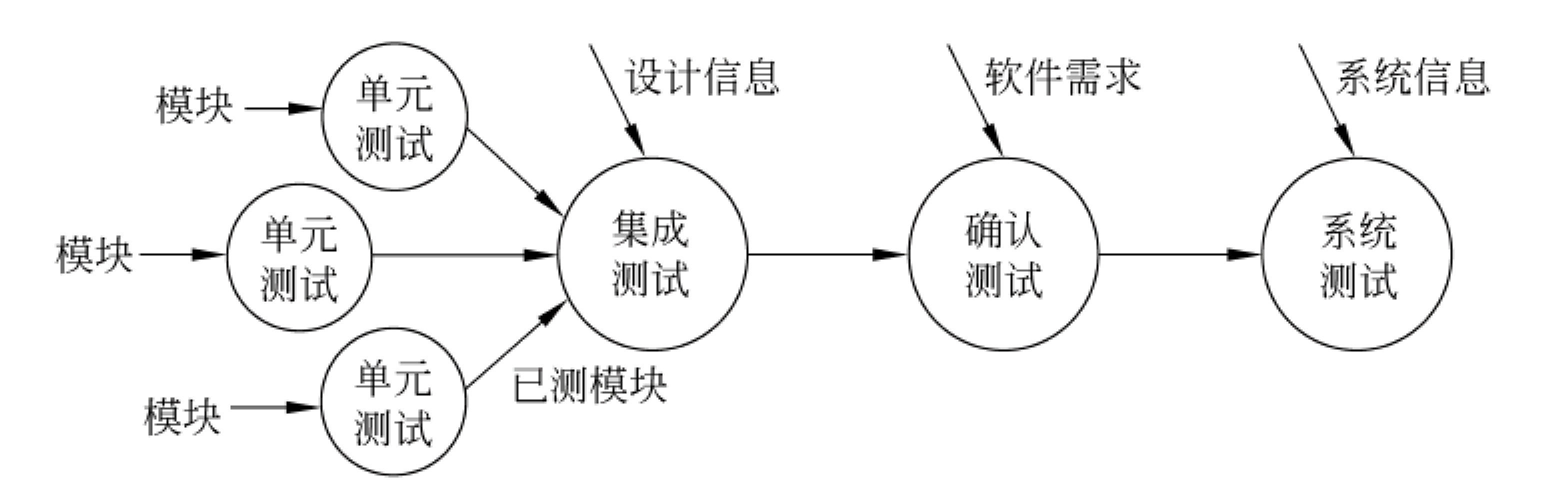


图 3-1　软件测试步骤

3.1　单元测试

单元测试又称为模块测试、逻辑测试、结构测试，单元测试是对软件基本组成模块和主要的控制路径进行的测试，进而发现模块内部的错误。有关测试复杂度和错误发现是受单元测试的约束范围限定的。单元测试采用白盒测试方法，而且可以多个单元并行进行测试。

3.1.1　单元测试内容

设计单元测试的测试用例时主要考虑的问题如图 3-2 所示。

1. 模块接口测试

模块接口测试检查进出程序单元的数据流是否正确。对模块接口数据流的测试必须在任何其他测试之前进行，如果不能确保数据正确地输入和输出，所有的测试都将无法进行，测试主要内容如下。

图 3-2　设计单元测试用例时要考虑的问题

(1) 形式参数与实际参数是否一致。

(2) 模块接收输入参数与模块变元是否一致。

(3) 模块调用内部函数变元个数、属性、次序是否一致。

(4) 文件的属性是否正确。

(5) 全程变量在各模块间的定义是否一致等。

2. 数据结构测试

在模块工作过程中，必须测试其内部的数据能否在算法执行的整个过程中保持完整性，包括内部数据的内容、形式及相互关系保证不发生错误。应该说模块的局部数据结构是经常发生错误的错误源，所以应该在单元测试中注意对局部数据结构的测试，测试内容如下。

(1) 变量从来没有被使用。

(2) 变量没有初始化。

(3) 错误的类型转换。

(4) 数组越界。

(5) 非法指针。

(6) 变量或函数名称拼写错误。

3. 边界条件测试

边界条件测试可保证模块在极限的情形下能够正确执行，测试内容如下。

(1) 普通合法数据是否能正确处理。

(2) 普通非法数据是否能正确处理。

(3) 边界内最接近边界的(合法)数据是否能正确处理。

(4) 边界外最接近边界的(非法)数据是否能正确处理。

4. 独立路径测试

在单元测试中，最主要的是路径测试。在控制结构中的所有独立路径(基本路径)都要走遍，进而保证在一个模块中的所有语句都能执行至少一次。在单元测试过程中，对执行路径的选择性测试是最主要的任务。测试用例必须能够发现由于计算错误、不正确的判定或不正常的控制流而产生的错误，测试内容如下。

(1) 死代码。

(2) 错误的计算优先级。

(3) 精度错误(比较运算错误、赋值错误)。

(4) 表达式的不正确符号(>、>=;=、==、!=)。

(5) 循环变量的使用错误(错误赋值)。

5. 出错处理测试

完善的模块设计要求能预见出错的条件,并设置适当的出错处理,以便在程序出错时,能对出错程序重新调整,保证其逻辑上的正确性。这种出错处理也是模块功能的一部分,测试内容如下。

(1) 检查错误是否出现,重点检查以下时刻。

① 资源使用前后。

② 其他模块使用前后。

(2) 出现错误,是否进行了错误处理,主要有以下几种处理方式。

① 抛出错误。

② 通知用户。

③ 进行记录。

(3) 错误处理是否有效,主要有以下两种判定方法。

① 在系统干预前处理。

② 报告和记录真实而详细描述了错误。

3.1.2 单元测试规则

单元测试与编码测试属于同一步骤。在代码开发、评审和语法正确性验证之后,单元测试用例设计就已开始。因为一个单元本身不是一个单独的程序,所以必须为每个单元测试开发驱动模块或/和桩模块软件。在大多数应用中,一个驱动模块只用于接收测试数据并把数据传送给要测试的单元。桩模块的功能是替代那些属于被测试单元的被调用模块。一个桩模块或空子程序使用从属模块的接口,可以做少量的数据操作,接收被测试模块的调用和输出数据,是被测试模块要调用的模块。

驱动模块和桩模块都是额外的开销,都属于必须开发但又不能和最终软件一起提交的软件,如果驱动模块和桩模块很简单的话,那么额外开销相对来说很低。但是,许多模块使用简单的额外软件不能进行足够的单元测试。在这些情况下,完整的测试要推迟到集成测试步骤时再完成。

当一个单元被设计为高内聚时,测试被简化。当一个单元只表示一个函数时,测试用例的数量就会降低,而且错误也就更容易被预测和发现,如图 3-3 所示。

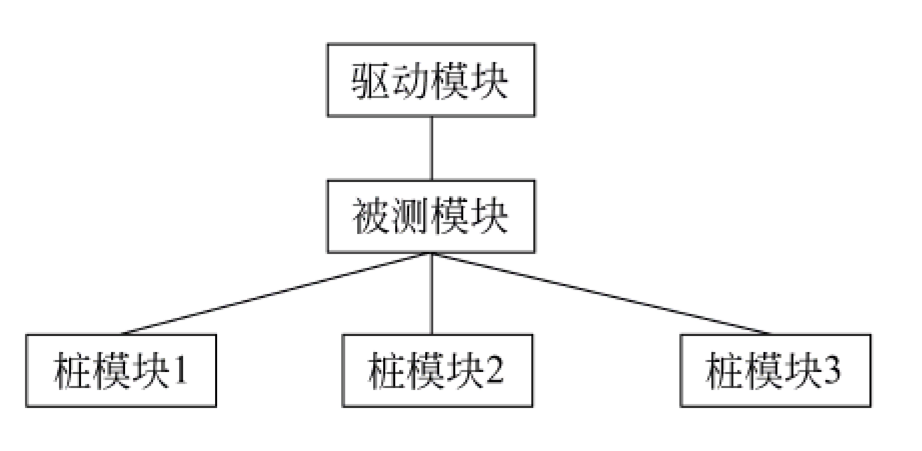


图 3-3 单元测试规则

1. 单元测试的步骤

(1) 配置测试环境,设计辅助测试模块、驱动模块和桩模块。

(2) 编写测试数据,根据单元测试要解决的问题设计测试用例。

(3) 可以进行多个单元的并行测试。

通过编译以后,首先做静态代码的复审。参加者为软件设计者、程序编写者和程序测试者,其中有软件设计能力很强的高级程序员任组长。在研究软件设计文档基础上召开审查会,分析程序逻辑与错误清单,经测试预演、人工测试、代码复审后再进入计算机代码执行活动的动态测试,再做单元测试报告。

为了提高测试效率,克服无法穷尽测试的困难,在单元测试中应采用白盒法与黑盒法相结合、静态测试与动态测试相结合、人工测试与机器测试相结合的策略。

2. 动态单元测试步骤

(1) 用边值分析方法设计测试集,测试边界易出错之处。

(2) 用等价类划分方法设计测试集,测试主要的软件错误。

(3) 结合用人工测试的错误推测法设计测试集做补充。

(4) 用逻辑覆盖设计测试集做补充。

3.1.3　单元测试的问题

1. 单元测试的重要性

(1) 覆盖了最小代码单元。

(2) 支持组包测试。

(3) 执行率是100%。

(4) 可以随时执行,并且覆盖变动后的代码范围,以确定变动后的代码对原代码功能未作修改。

(5) 提升软件系统整体信赖度。

(6) 并不仅仅体现在测试覆盖上,还包含对可能出现问题的代码进行彻底排查。

(7) 支持先测试后编码的行为。

(8) 支持变化,因为任何变化所导致的失败情况都会被立刻反映出来。

(9) 便于后期维护,因为测试更能准确反映原代码设计人员的思维。

2. 单元测试的误区

1) 集成测试将找出所有的Bug

由于规模大的代码集成存在较高的复杂性。如果软件单元没有进行测试,开发人员很可能花费大量的时间仅仅是为了使软件能够运行,而任何实际的测试方案都无法执行。一旦软件可以运行了,开发人员又要面对这样的问题:在考虑软件全局复杂性的前提下对每个单元进行全面的测试。这是一件非常困难的事情,甚至在创造一种单元调用的测试条件的时候,要全面地考虑单元被调用时的各种入口参数。在软件集成阶段,对单元功能进行全面测试的复杂程度远远超过独立进行的单元测试过程。

最后的结果是测试将无法达到它所应该有的全面性。一些缺陷将被遗漏,并且很多

Bug 将被忽略。

2）单元测试浪费了过多时间

编码一旦完成，很多人迫不及待地进行软件集成工作，以便及早看到运行结果。此时进行单元测试，错误地认为单元测试浪费时间，进而成为障碍。

在开发过程中，难免会有代码错误，而使得软件无法运行或者无法正常运行，修改错误成为软件开发新增的额外工期，而且当这个系统投入使用时也无法确保它能够可靠地运行。而进行完整的单元测试和编写实际的代码所花费的精力大致相同。一旦完成了单元测试，则可以进行更高效的集成工作，所以单元测试是对项目可用时间更高效的利用。

3）可以不进行单元测试

许多程序员认为自己开发的软件能直接正常运行，测试是浪费时间。而忽视了编码不是一个可以一次性通过的过程，人们所认为的正确性局限在人们的思维中，或者局限在某个时间段中，当这种正确性因为现实环境的变化被否定时，错误就产生了，所以单元测试是软件测试中必不可少的一步。

3.2 集成测试

如果每一个模块都能正常工作，但把它们集成到一起不能正常工作，这主要就是由于模块相互调用时的接口处引入许多新问题。例如，数据经过接口时可能丢失；一个模块对另一模块可能造成不应有的影响；几个子功能组合起来不能实现主功能；误差不断积累达到不可接受的程度；全局数据结构出现错误，等等。集成测试是通过测试发现与接口有关的问题来构造程序的系统化技术，其目标是利用通过了单元测试的模块，构造设计中所描述的程序结构。

非增量集成是指把所有模块全部集成起来，进行整体测试。然而，这样做的结果是测试时可能发现一大堆错误，为每个错误定位和纠正非常困难，并且在改正一个错误的同时又可能引入新的错误，新旧错误混杂，更难断定出错的原因和位置。

增量式集成是与之相反的方法，程序段段地扩展，测试的范围步步地增大，错误易于定位和纠正，界面的测试可做到完全彻底。在集成测试过程中，首先把已经通过单元测试的模块组装起来，构成一个在设计阶段所定义的程序结构，然后通过集成测试发现与接口有关的问题。

1. 集成测试的重要性

（1）尽管开发时尽力做到仔细，但是并不能保证接口不出错。也就是说虽然可以通过单元测试验证单个或者几个模块之间工作正常，但是并不能保证所有模块组合起来就能正常工作。

（2）在单元测试中用了很多桩模块，但并不能保证这些模块在实际的系统中的真实性。

（3）所有模块出错的概率不同，集成测试中可以找出那些容易使别的模块出错的模块。

（4）能尽早发现错误。

（5）报告错误后，可同时进行 Bug 消除和开发工作。

2. 集成测试的主要过程

(1) 构建的确认过程。

(2) 补丁的确认过程。

(3) 测试组提交过程。

(4) 测试用例设计过程。

(5) 测试代码编写过程。

(6) Bug 的报告过程。

(7) 每周/每两周的构建过程。

(8) 点对点的测试过程。

(9) 组内培训过程。

集成测试的方案有很多种,如自顶向下集成测试、自底向上集成测试、混合式集成测试、核心系统先行集成测试、高频集成测试和回归测试等。

3.2.1　自顶向下集成测试

自顶向下集成是构造程序结构的一种增量式方式,它从主控模块开始,按照软件的控制层次结构,以深度优先或宽度优先的策略,逐步把各个模块集成在一起。深度优先策略首先把主控制路径上的模块集成在一起,至于选择哪一条路径作为主控制路径,一般根据问题的特性确定。以图 3-4 为例,若选择了最左一条路径,首先将模块 M1、M2、M5 集成在一起,再将 M5 或 M6 集成起来,然后考虑中间和右边的路径。宽度优先策略则不然,它沿控制层次结构水平地向下移动。以图 3-4 为例,它首先把 M2、M3 和 M4 与主控模块集成在一起,再将 M5 和 M6 集成起来,然后继续。

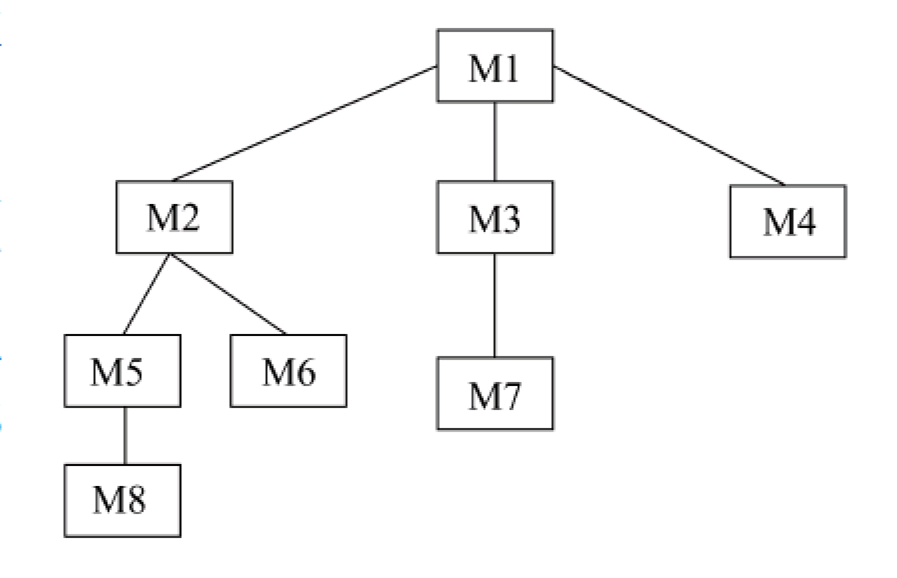


图 3-4　自顶向下集成测试

自顶向下集成测试的步骤如下。

(1) 以主控模块作为测试驱动程序,把对主控模块进行单元测试时引入的所有桩模块用从属于主控模块的模块来替代。

(2) 依据所选的深度优先或宽度优先集成策略,每次只替代一个桩模块。

(3) 每集成一个模块立即测试一遍。

(4) 每完成一次测试,接下来就有一个桩模块被一个真正的模块所替代。

(5) 为避免引入新错误,可以不断地进行回归测试,即全部或部分地重复已做过的测试。

从(2)步开始,循环执行上述步骤,直至整个程序构造完毕。

在图 3-4 中,实线表示已部分完成的结构,若采用深度优先策略,下一步将用模块 M7 替换桩模块 S7,当然 M7 本身可能又带有桩模块,随后将被对应的实际模块替代。

自顶向下集成测试的优点在于能尽早地对程序的主要控制和决策机制进行检验,因此较早地发现错误。缺点是在测试较高层模块时,低层处理采用桩模块替代,不能反映真

实情况,重要数据不能及时回送到上层模块,因此测试并不充分。解决这个问题有下述几种办法。

(1) 把某些测试推迟到用真实模块替代桩模块之后进行。

(2) 开发能模拟真实模块的桩模块。

(3) 自底向上集成模块。

方法(1)又回退为非增量式的集成方法,使错误难于定位和纠正,并且失去了在组装模块时进行一些特定测试的可能性;方法(2)要大大增加开销;方法(3)比较切实可行,下面介绍自底向上集成测试。

3.2.2 自底向上集成测试

自底向上测试是从最低层模块开始组装测试,因测试到较高层模块时,所需的下层模块功能均已具备,所以不再需要桩模块。自底向上集成测试的步骤如下。

(1) 把低层模块组织成实现某个特定子功能的模块群。

(2) 开发一个测试驱动模块,控制测试数据的输入和测试结果的输出。

(3) 对每个模块群进行测试。

(4) 删除测试使用的驱动模块,用较高层模块把模块群组织成为完成更大功能的新模块群。

从(1)步开始循环执行上述各步骤,直至整个程序构造完毕。

如图 3-5 所示,首先将最低层模块分为 3 个模块群,每个模块群引入一个驱动模块进行测试。因模块群 1、模块群 2 中的模块均属于模块 Ma,因此在驱动模块 D1、D2 去掉后,模块群 1 与模块群 2 直接与 Ma 相连。以此类推,模块群 3 属于模块 Mb,把驱动模块 D3 去掉后,Mb 与模块群 3 直接连接,最后 Ma、Mb 和 Mc 全部集成在一起进行测试。

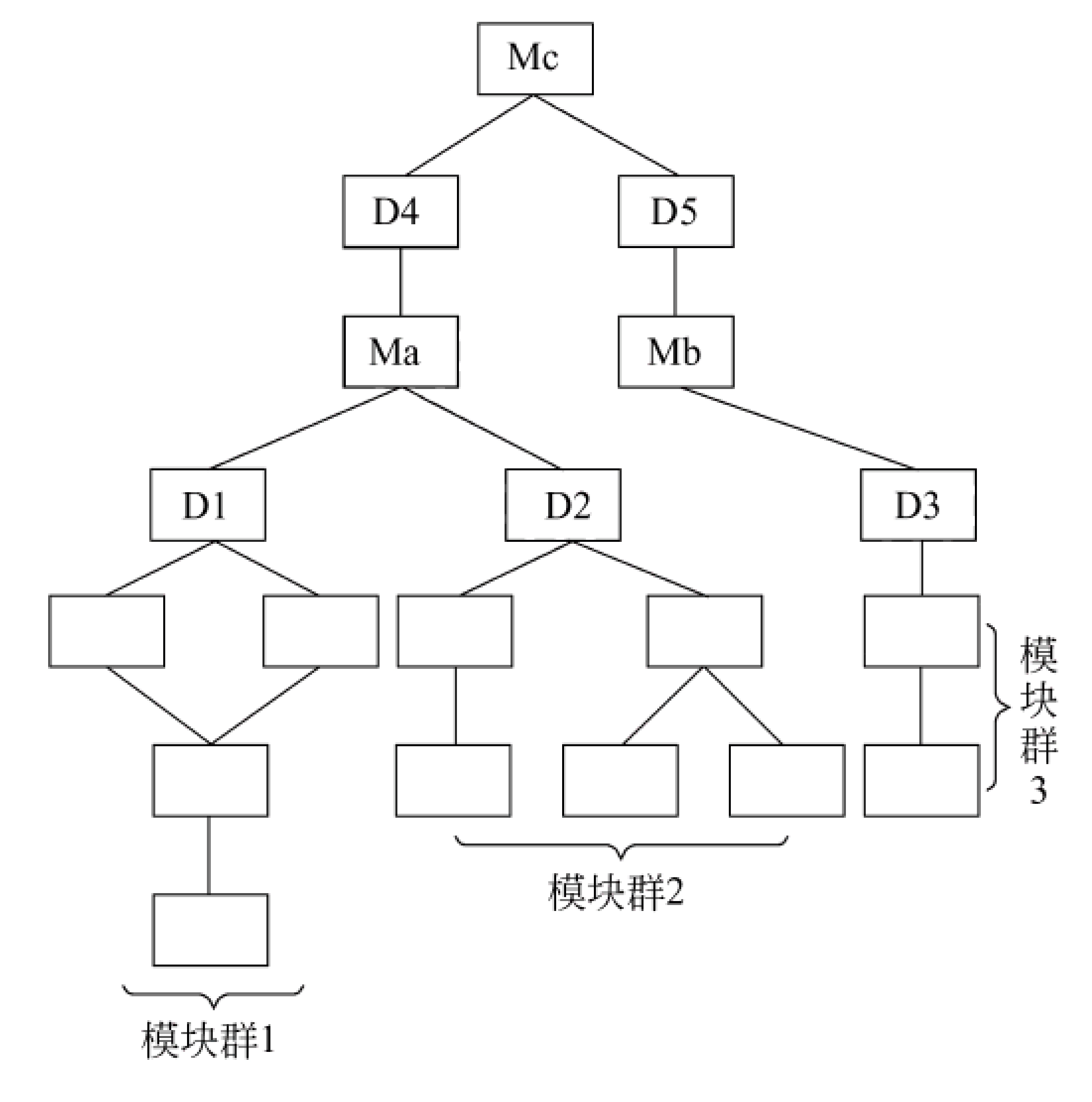


图 3-5 自底向上集成测试

自底向上集成测试的方法是最常用的测试方法。自底向上集成测试的优点是：不用桩模块，管理方便、测试人员能较好地锁定软件故障所在位置，测试用例的设计相对简单，但缺点是程序最后一个模块加入时才具有整体形象，对于某些开发模式不适用，如使用XP开发方法，它会要求测试人员在全部软件单元实现之前完成核心软件部件的集成测试。它与自顶向下集成测试方法优缺点正好相反。因此，在测试软件系统时，应根据软件的特点和工程的进度，选用适当的测试策略，有时混合使用两种策略更为有效，上层模块用自顶向下的方法，下层模块用自底向上的方法。

在集成测试中要注意关键模块，关键模块一般都具有下述特征。

（1）对应几条需求。

（2）具有高层控制功能。

（3）复杂、易出错。

（4）有特殊的性能要求。

关键模块应尽早测试，并反复进行回归测试。

3.2.3　混合式集成测试

混合式集成测试是自顶向下集成测试与自底向上集成测试的折中测试方法，结合了两者的优点。混合式集成测试把系统划分成3个层次，中间一层作为目标层。测试的时候，对目标层的上一层使用自顶向下集成策略，对目标层下一层使用自底向上的集成策略，最后测试都结合到目标层。

如图3-6所示，其中目标层为B、C、D。目标层上面一层是A，目标层下面一层是E、F、G。其具体步骤如下。

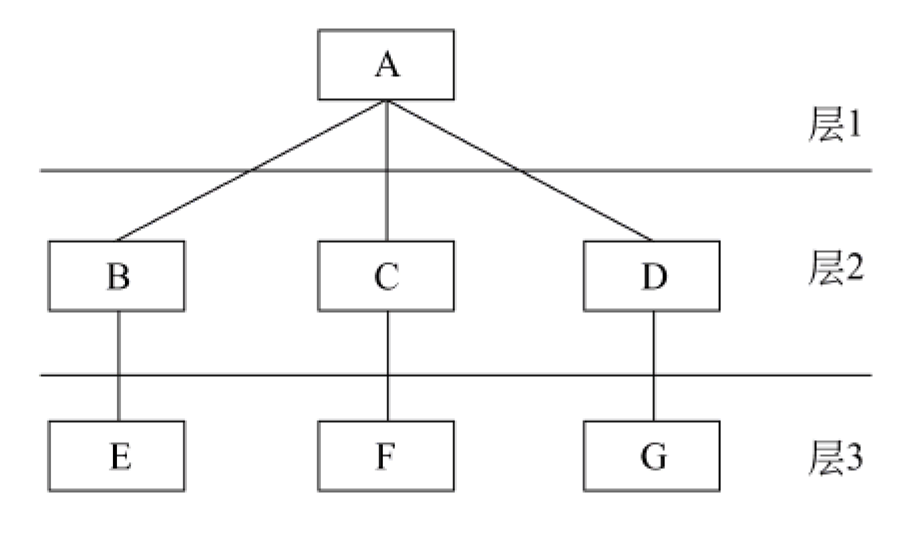


图3-6　混合式集成测试

（1）对目标层的上一层使用自顶向下的集成策略，因此测试A，使用桩模块来代替B、C、D。

（2）对目标层的下一层使用自底向上的集成策略，因此测试E、F、G，使用驱动模块来代替B、C、D。

（3）把目标层的下面一层与目标层集成，因此测试(B，E)、(C，F)、(D，G)，使用驱动模块来代替A。

（4）把三层都集成起来，因此测试(A，B，C，D，E，F，G)。

在进行混合式集成测试时，要尽量减少驱动模块和桩模块的数量。因此，在上面集成时先使用目标层与下面的一层结合来进行测试，而不是用与上面的一层结合来进行测试，因为先与上面集成时增加了下面桩模块的设计。

3.2.4　先行集成测试

先行集成测试法的思想是先对核心软件部件进行集成测试，在测试通过的基础上再

按各外围软件部件的重要程度逐个集成到核心系统中。每次加入一个外围软件部件都产生一个产品基线，直至最后形成稳定的软件产品。核心系统先行集成测试法对应的集成过程是一个逐渐趋于闭合的螺旋形曲线，代表产品逐步定型的过程。

核心系统先行集成测试的步骤如下。

(1) 对核心系统中的每个模块进行单独的、充分的测试，必要时使用驱动模块和桩模块。

(2) 对于核心系统中的所有模块一次性集合到被测系统中，解决集成中出现的各类问题。在核心系统规模相对较大的情况下，也可以按照自底向上的步骤，集成核心系统的各组成模块。

(3) 按照各外围软件部件的重要程度以及模块间的相互制约关系，拟定外围软件部件集成到核心系统中的顺序方案。方案经评审以后，即可进行外围软件部件的集成。

(4) 在外围软件部件添加到核心系统以前，外围软件部件应先完成内部的模块级集成测试。

(5) 按顺序不断加入外围软件部件，排除外围软件部件集成中出现的问题，形成最终的用户系统。

该集成测试方法对于快速软件开发很有效果，适合较复杂系统的集成测试，能保证一些重要的功能和服务的实现。缺点是采用此法的系统一般应能明确区分核心软件部件和外围软件部件，核心软件部件应具有较高的耦合度，外围软件部件内部也应具有较高的耦合度，但各外围软件部件之间应具有较低的耦合度。

3.2.5 高频集成测试

高频集成测试是指同步于软件开发过程，每隔一段时间对开发团队的现有代码进行一次集成测试。如某些自动化集成测试工具能实现每日深夜对开发团队的现有代码进行一次集成测试，然后将测试结果发到各开发人员的电子邮箱中。该集成测试方法频繁地将新代码加入到一个已经稳定的基线中，以免集成故障难以发现，同时控制可能出现的基线偏差。使用高频集成测试需要具备一定的条件：可以持续获得一个稳定的增量，并且该增量内部已被验证没有问题；大部分有意义的功能增加可以在一个相对稳定的时间间隔(如每个工作日)内获得；测试包和代码的开发工作必须是并行进行的，并且需要版本控制工具来保证始终维护的是测试脚本和代码的最新版本；必须借助于使用自动化工具来完成。高频集成测试一个显著的特点就是集成次数频繁，显然，人工的方法是不能胜任的。高频集成测试步骤如下。

(1) 选择集成测试自动化工具。如很多 Java 项目采用 JUnit+Ant 方案来实现集成测试的自动化，也有一些商业集成测试工具可供选择。

(2) 设置版本控制工具，以确保集成测试自动化工具所获得的版本是最新版本。如使用 CVS 进行版本控制。

(3) 测试人员和开发人员负责编写对应程序代码的测试脚本。

(4) 设置自动化集成测试工具，每隔一段时间对配置管理库新添加的代码进行自动化的集成测试，并将测试报告汇报给开发人员和测试人员。

(5) 测试人员监督代码开发人员及时关闭不合格项。

按照(3)至(5)不断循环,直至形成最终软件产品。

高频集成测试能在开发过程中及时发现代码错误,能直观地看到开发团队的有效工程进度。高频集成测试中,开发维护源代码与开发维护软件测试包被赋予了同等的重要性,这对有效防止错误、及时纠正错误都很有帮助。它的缺点在于测试包有时候可能不能暴露深层次的编码错误和图形界面错误。

3.2.6 回归测试

在集成测试中,每当一个新的模块加进来的时候,软件就发生了改变。新的数据流路径被建立,新的操作可能也会出现,还有可能激活了新的控制逻辑。这些改变可能会使原来工作很正常的功能产生错误。在集成测试策略的环境中,回归测试是对某些已经进行过的测试的子集的重新执行,以保证上述改变不会有副作用。

成功测试的结果是发现错误,而错误是要被修改的,每当软件被修改的时候,软件配置的某些方面也被修改了,回归测试就是保证改动不会带来不可预料的行为或者附加的错误。

回归测试可以通过重新执行所有的测试用例的一个子集人工地进行,也可以使用自动化的捕获/回放工具来进行。

回归测试集包括下述 3 种类型的测试用例。

(1) 能够测试软件的所有功能的代表性测试用例。

(2) 专门针对可能会被修改影响的软件功能的附加测试。

(3) 注重于修改过的软件构件的测试。

在集成测试进行的过程中,回归测试可能会变得非常庞大。因此,回归测试应只包括涉及在主要的软件功能中出现的一个或多个错误类的那些测试。每进行一个修改时,就对每一个程序功能都重新执行所有的测试是不实际的,而且效率也低。

3.3 确认测试

集成测试完成以后,分散开发的模块被连接起来,构成完整的程序。各模块之间的接口存在的问题都已消除,于是测试工作进入确认测试阶段。确认测试主要由两部分组成:有效性测试和配置复审。

有效性测试主要检查软件是否达到了系统设计要求,是否满足软件需求说明书中的确认标准,若能达到,则认为开发的软件是合格的。有效性测试主要是通过一系列黑盒测试来进行的,其测试用例来自于需求分析阶段。

确认测试的另一个重要部分是配置复审,复审的目的在于保证软件配置齐全、分类有序,并且包括软件维护所必需的细节。

确认测试的步骤如下。

(1) 在模拟的环境中进行强度测试,在事先规定时期内运行所有软件功能,发现软件

与原目标不符的错误。

(2) 执行测试计划中提出的所有确认测试。

(3) 测试用户手册和操作手册，证实其实用性与有效性，并改正其中的错误。

(4) 分析测试结果，找出产生错误的原因。

(5) 书写确认测试的分析报告。

(6) 确认测试结束，书写整个项目的开发总结报告。

(7) 整理所有文件。

(8) 评审。

(9) 邀请用户参加测试。

(10) 交付的文件有：确认测试的分析报告、用户手册、操作手册、项目开发总结报告。

3.3.1 确认测试的标准

通过一系列黑盒测试来实现软件确认。确认测试需要制定测试计划和过程，测试计划应规定测试的种类和测试进度，测试过程需要定义一些特殊的测试用例，通过测试和获得的结果来说明被测软件与需求是否一致。无论是计划还是过程，都应该着重考虑软件是否满足合同规定的所有功能和性能，文档资料是否完整、准确，人机界面、可移植性、兼容性、错误恢复能力和可维护性等是否令用户满意。

确认测试有两种可能的结果，一种结果是测试结果与预期的结果相符，这说明软件的功能和性能特征与需求规格说明书相符，用户可以接受；另一种结果是测试结果与预期的结果不符，这说明软件的功能和性能特征与需求规格说明书不一致，需要提交一份问题报告，进行到这个阶段才发现严重错误和偏差将很难在预定的工期内改正，因此必须与用户协商，找到可接受的解决问题的方法。

3.3.2 有效性测试

有效性测试是在模拟的环境下，运用黑盒测试方法，确认组装完毕的程序是否满足软件需求规格说明书列出的要求。

在有效性测试中，除了考虑功能和性能的需求以外，还需检验其他方面的需求，如可移植性、兼容性、用户界面以及系统所提供的文档资料是否符合要求等。

1. 功能性测试需求

功能性测试需求来自于测试对象的功能性说明。对于需求规格说明书中的功能描述，将至少派生一个测试需求。

2. 性能测试需求

性能测试需求来自于测试对象的指定性能行为。性能通常被描述为对响应时间和资源使用率的评测。性能需要在各种条件下进行评测，这些条件包括以下一些。

(1) 不同的工作量和/或系统条件。

(2) 不同的用例或功能。

(3) 不同的配置。

3. 其他测试需求

其他测试需求包括配置测试、安全性测试、容量测试、强度测试、故障恢复测试、负载测试等，测试需求可以从非功能性需求中发现与其对应的描述。每一个描述信息可以生成至少一个测试需求。

3.3.3　配置复审

配置复审也是确认测试的重要环节。其目的在于确保已开发软件的所有文档资料均已编写齐全，足以支持投入运行以后的软件维护工作。这些文档资料包括用户所需资料（如用户手册、操作手册等）、设计资料（如设计说明书等）、源程序以及测试资料（如测试说明书、测试报告等）。

图 3-7 详细说明了配置复审与有效性测试的关系。

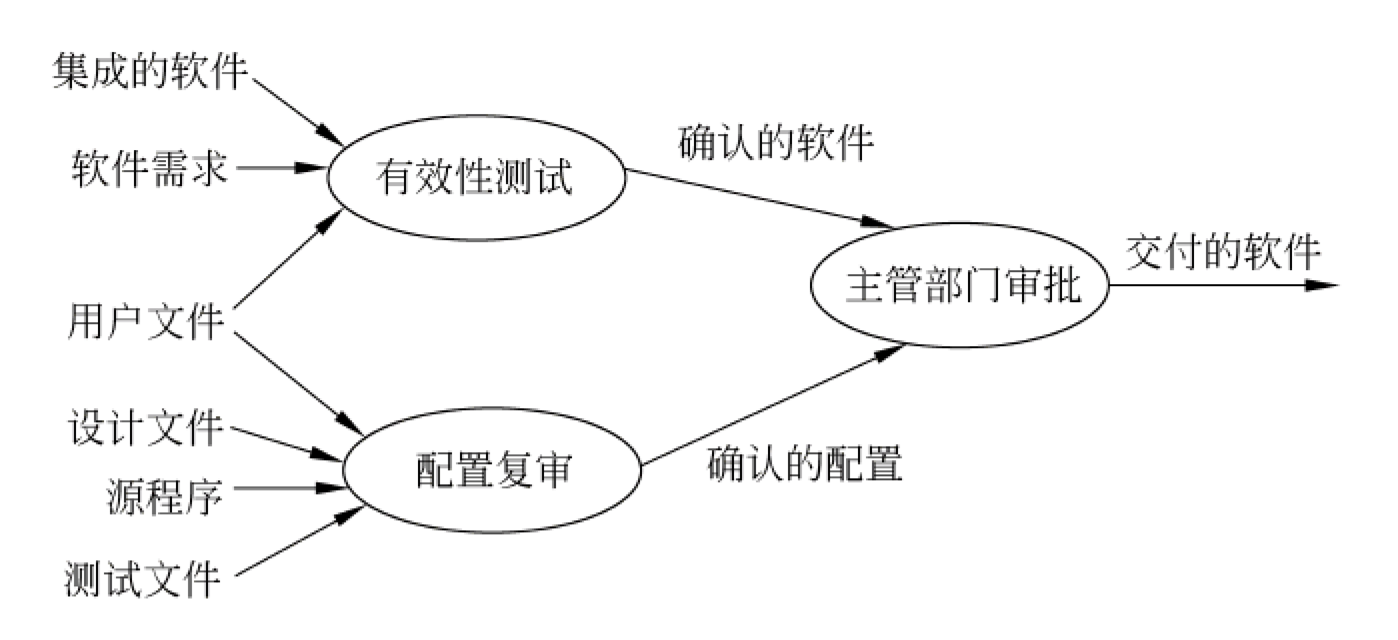


图 3-7　配置复审与有效性测试的关系

3.3.4　α 测试与β 测试

软件开发人员不可能完全预见用户实际使用程序的情况。这是由于用户可能错误地理解了命令，或提供一些莫名其妙的数据组合，或对输出信息迷惑不解等。软件是否真正满足最终用户的要求，应由用户进行一系列验收测试。验收测试既可以是非正式的测试，也可以是有计划的测试。有时验收测试长达数周甚至数月，不断暴露出的错误导致了开发延期。一个软件产品，可拥有众多用户，不可能由每个用户一一验收，此时可采用 α 测试与 β 测试，可以发现那些只有最终用户才能发现的问题。

1. 验收测试

由于软件是为用户而开发的，则需要进行一系列的验收测试来保证满足用户所有的需求。验收测试是以用户为主的测试，软件开发人员和质量保证人员也应参加测试，并由用户参加设计测试用例和分析测试的输出结果。

2. α 测试

α 测试是指软件开发公司组织内部人员模拟各类用户对即将完成的软件产品（称为 α 版本）进行测试。它是在开发环境下，由软件开发公司组织内部人员在开发者的指导下进行的测试。被测试的软件是在开发者可以控制的环境中进行的，并由开发者负责记录

错误和使用中出现的问题。α 测试的关键在于尽可能逼真地模拟实际运行环境和用户对软件产品的操作,并尽最大努力涵盖所有可能的用户操作方式。它测试的是评价软件产品的 FLURPS。FLURPS 是功能、本地化、可使用性、可靠性、性能和支持的英文缩写,是软件产品性能和功能的综合描述,尤其注重产品的界面和特色。α 测试人员是除产品开发人员之外首先见到产品的人,他们提出的功能和修改意见特别有价值。α 测试可以从软件产品编码结束之时开始,或在模块(子系统)测试完成之后开始,也可以在确认测试过程中产品达到一定的稳定和可靠程度之后再开始。有关的手册等文档资料应事先准备好。

3. β 测试

1) β 测试的基本概念

经过了 α 测试的软件产品称为 β 版本,紧随其后的 β 测试是指软件开发公司组织各方面的典型用户在日常工作中实际使用 β 版本,并要求用户报告异常情况,提出批评意见,然后软件开发公司再对 β 版本进行修改和完善。

β 测试不同于 α 测试,是由用户在实际使用环境下进行测试的。被测试的软件是在开发者无法控制的环境下进行的,β 测试是对软件进行的实际应用性检验。所有在测试中遇到的问题均由用户记录,并定期把它们报告给开发者,开发者在接收到 β 测试的问题报告之后,对系统进行最后的修改,然后就开始准备向所有的用户发布最终的软件产品。β 测试主要衡量产品的 FLURPS,着重于产品的支持性,包括文档、客户培训和支持产品生产能力。只有当 α 测试达到一定的可靠程度时,才能开始 β 测试,它处在整个测试的最后阶段。同时,产品的所有手册文本也应该在此阶段完全定稿。

由于 β 测试的主要目标是测试可支持性,所以 β 测试应尽可能由主持产品发行的人员来管理。

2) β 测试的优点

(1) 参与者可以比别人先一步看到软件的新特性。

(2) 参与者可以发现一些值得怀疑的错误。

(3) 检测出那些出现错误的结果,将使软件变得更好。

(4) 通过 β 测试人员的反馈,将影响以后开发的方式。

3) 非传统的 β 测试

β 测试作为一种测试方式,存在着多种不同的 β 测试方法,根据不同的需求,衍生了一些非传统的 β 测试。以下简单介绍一些非传统的 β 测试。

(1) 产品发布后的 β 测试。β 测试通常都是针对发布到市场之前的产品。其主要思想是在产品发布之前找出产品存在的问题。但在某些情况下,公司可能会决定先把产品发布到市场上,然后当产品已经出现在柜台上时再进行 β 测试。

(2) 公开 β 测试。β 测试通常都着重于为产品的发布作准备。但是当某种产品有这种要求,或需要进行大规模测试的重要性远远超过了公开发布本身时,就需要进行公开 β 测试。

(3) 小型 β 测试。小型 β 测试是一种快速而不公正的测试,它只关注于迅速地得到测试结果。这种测试通常只有少量的测试人员参加,而且测试周期也比较短。决定执行

小型β测试的公司通常都很清楚β测试在产品开发过程中的价值，但却没有足够的时间和人力去实行常规的β测试过程。

(4) 集中β测试。集中测试的含义正如其名称所表述的那样。β测试作为产品的一个组成部分，所有测试成员的精力都集中于此处。通常，这种测试可以用来证实一个概念、验证某个问题或者强调某一特性的测试。总体来说，这种β测试十分有效，但是，测试人员也很快就会精疲力竭。集中测试总是强调某一特定问题，动员所有测试人员将工作重心转移到该目标上。一旦这个目标实现了，测试人员就会停下来，等待转向新的方向。

(5) 内部β测试。当公司急于发布产品时，公司认为执行β测试的最好方式就是把产品分发给自己的员工，让他们模拟用户环境进行测试，然后向公司报告测试结果。但是，公司员工不是目标市场，所以他们对产品的观点会受到影响并存有偏见。

3.4　系统测试

软件只是计算机系统和计算机应用系统的一个元素，软件最终要与其他元素(如硬件、外部设备、某些支持软件、信息、数据等)相结合，进行集成测试和确认测试，以保证系统能在一定的硬件和软件环境下正常运行。

系统测试的目的是将系统已实现的功能与其设计目标进行比较，发现系统与系统定义不符合或不一致的地方。系统测试通常是在真实的环境下所进行的测试，系统测试的用例应该根据需求分析说明书来设计，并在实际使用环境下来运行，系统测试也只能采用黑盒测试技术，系统测试的内容包括：软件环境、硬件环境、业务系统的安装和配置、业务系统的运行。系统测试主要是对系统的正确性和完整性所进行的测试。系统测试不仅由软件开发人员来完成，而且需要系统各个方面人员的参与。

在系统测试之前，应完成下列工作。

(1) 为测试软件系统的输入信息设计出错处理通路。

(2) 设计测试用例，模拟错误数据和软件界面可能发生的错误，记录测试结果，为系统测试提供经验和帮助。

(3) 参与系统测试的规划和设计，保证软件测试的合理性。

3.4.1　系统测试的种类

系统测试由多种测试组成，目的是充分运行系统，验证系统各部件是否都能正常工作并完成任务。系统测试的种类主要包括以下一些。

- 恢复测试：验证系统在经历崩溃、硬件故障或类似的毁坏性设置后，能否恢复到可用状态。
- 安全测试：验证只有授权的用户可以允许访问的特性。
- 强度测试：强迫系统在不合理的负载下操作，但不给系统提供处理负载所需要的资源。目的是测试系统在超出规定的时间内执行操作的能力。
- 性能测试：度量在高峰和正常条件下执行任务所花费的时间，以确保系统在特定

的时间约束下正常响应。

- 功能测试：验证每个功能是否符合需求。
- 负载测试：在短时期内让有经验的用户充分使用系统，目的是模拟实际环境的交互，或者在一定的负载下测试应用程序，以判断系统在何种情况下发生响应时间降级或失败。
- 适用性测试：确保任何内部维护信息（如跟踪和诊断消息）与文档一致。
- 配置测试：使用系统必须支持的不同软件和硬件配置，例如不同的外部设备。
- 兼容性测试：评价软件在由硬件、软件、操作系统和网络的特定版本所定义的特定环境中执行情况如何。
- 安装测试：验证完全、部分或升级安装过程与文档一致，此外还要测试卸载过程。
- 可靠性测试：验证系统在指定的条件和规定的时间内执行操作的能力。
- 可用性测试：评价应用程序的用户友好程度，并找出对用户难度较大的操作，包括验证手工操作及面向用户的程度。

几类较典型的系统测试介绍如下。

1. 恢复测试

许多计算机系统必须在一定的时间内从错误中恢复过来，然后继续运行。在有些情况下，一个系统必须是可以容错的，这就是说，运行过程中的错误不能使得整个系统的功能都停止。在其他情况下，一个系统错误必须在一个特定的时间段之内改正，否则就会产生严重的经济损失。

恢复测试主要检查系统的容错能力，是指当系统出错时，能否在指定时间间隔内修正错误并重新启动系统。恢复测试是要采取各种人工干预方式强迫系统出错，而不能正常工作，进而检验系统的恢复能力。对于系统本身能够自动恢复的，需验证重新初始化、数据恢复、重新启动、检验点设置是否正确；对于人工干预的恢复系统，还需估测平均修复时间，确定其是否在可接受的范围内。

2. 安全测试

任何敏感信息或者能够对个人造成伤害的计算机系统，都是非法侵入的目标。侵入范围包括：只是为练习而试图侵入系统的黑客；为了报复而试图攻破系统的雇员；还有为了得到非法的利益而试图侵入系统的个人；等等。

安全测试的目的在于验证安装在系统内的保护机构确实能够对系统进行保护，使之不受各种不正常的干扰。系统的安全测试要设置一些测试用例，试图突破系统的安全保密措施，检验系统是否有安全保密的漏洞。安全测试期间，测试人员可以假扮非法入侵者，采用各种办法试图突破防线。例如，①想方设法截取或破译口令；②专门定做软件破坏系统的保护机制；③故意导致系统失败，企图趁系统恢复之机非法进入；④试图通过浏览非保密数据，推导所需信息；等等。在理论上，只要有足够的时间和资源，没有不可进入的系统。系统设计者的任务就是，把系统设计为攻破系统而付出的代价大于攻破系统之后得到的信息的价值。此时非法侵入者已无利可图，进而增强了系统的安全性。

3. 强度测试

强度测试的目的是要对付非正常的情形。强度测试通过设计测试用例，来检验系统的能力最高能达到什么实际的限度，让系统处于资源的异常数量、异常频率、异常批量的条件下运行，进而测试系统的承受能力。一般取比平常限度高 5～10 倍的限度做测试用例。例如，①如果正常的中断平均频率为每秒 1～2 次，强度测试设计为每秒 10 次中断；②某系统正常运行可支持 10 个终端并行工作，强度测试则检验 15 个终端并行工作的情况等。从本质上来说，测试者是通过破坏程序的运行来检验程序的承受能力。图 3-8 所示的是一种压力测试用例的参考模板。

1. 被测试对象的介绍
2. 测试范围与目的
3. 测试环境与测试辅导工具的描述
4. 测试驱动程序的设计
5. 压力测试用例

极限名		
前提条件		

图 3-8　压力测试用例参考模板

敏感测试是强度测试的一个演变。在有些情形下，在有效数据界限之内的一个很小范围的数据可能引起极端的甚至是错误的运行，或者引起性能的急剧下降。敏感测试就是要发现在有效数据输入中的可能引发不稳定或者错误处理的数据组合。

强度测试主要是研究系统在短时间内活动处在峰值时的反应。强度测试应在开发过程中尽早进行，可以发现主要的设计缺陷，这些缺陷将影响到多数领域，如果不尽早进行强度测试，一些早期的明显缺陷则难以发现。

4. 性能测试

1）基本概念

性能测试是用来测试软件在集成系统中的运行性能的。性能测试一是为了检验性能是否符合需求，二是为了得到某些性能数据供人们参考。有时关心测试的绝对值，如数据传输速率是每秒多少比特，有时关心测试的相对值，如某个软件比另一个软件快多少倍。在获取测试的绝对值时，要充分考虑并记录运行环境对测试的影响。例如计算机主频、总

线结构和外部设备都可能影响软件的运行速度;若多个计算机共享资源,软件运行可能很慢。在获取测试的相对值时,要确保被测试的几个软件运行于完全一致的环境中。硬件环境的一致性比较容易做到(用同一台计算机即可)。但软件环境的因素较多,除了操作系统,程序设计语言和编译系统对软件的性能也会产生较大的影响。如果是比较几个算法的性能,就要求编程语言和编译器也完全一致。

性能测试可发生在测试过程的所有步骤中,即使是在单元层,一个单独模块的性能也可以使用白盒测试来进行评估。然而,只有当整个系统的所有成分都集成到一起之后,才能检测出一个系统的真正性能。

性能测试经常和强度测试一起进行,而且常常需要硬件和软件测试设备,也就是说,在一种苛刻的环境中衡量资源的使用常常是必要的。外部的测试设备可以监测执行的间歇,当出现情况(如中断)时记录下来。通过对系统的检测,测试者可以发现导致效率降低和系统故障的原因。

2) 软件性能研究方面的基本步骤

(1) 文档化性能目标。

(2) 定义测试驱动或者用于驱动系统的输入源。

(3) 定义要使用的性能测试方法或工具。

(4) 定义性能研究如何被进行。

(5) 定义报告过程。

3) 注意事项

(1) 计算机的运算很快,通常人们来不及反应就结束了,应当编写一段程序用于计算时间以及相关数据。

(2) 测试软件在标准配置和最低配置下的性能。

(3) 不仅要记录软件的硬件环境,还要记录多用户并发工作情况。

(4) 为了排除干扰,应当关闭那些消耗内存,占用 CPU 的其他应用软件。

(5) 系统要测试的性能的种类比较多,应当分别赋予唯一的名称,切勿笼统地用性能两字。例如文档管理软件的性能种类有“文件上载速度”、“文件下载速度”等。

(6) 不同的输入情况将得到不同的性能数据,应当分档记录。例如传输文件的容量从 100KB 到 1MB 可以分成若干等级。

(7) 由于环境的波动,同一输入情况在不同的时间可能得到不同的性能数据,可以取其平均值。

3.4.2 系统测试与单元测试、集成测试之间的区别

1. 测试方法不同

系统测试属于黑盒测试;单元测试、集成测试属于白盒测试或灰盒测试。

2. 测试范围不同

单元测试主要测试模块的内部接口、数据结构、逻辑、异常处理等对象;集成测试主要

测试模块之间的接口和异常；系统测试主要测试整个系统是否满足用户的需求。

3. 评估基准不同

系统测试的评估基准是测试用例对需求规格的覆盖率；单元测试和集成测试的评估主要是代码的覆盖率。

3.4.3　系统测试的位置

系统测试的位置如图 3-9 所示。

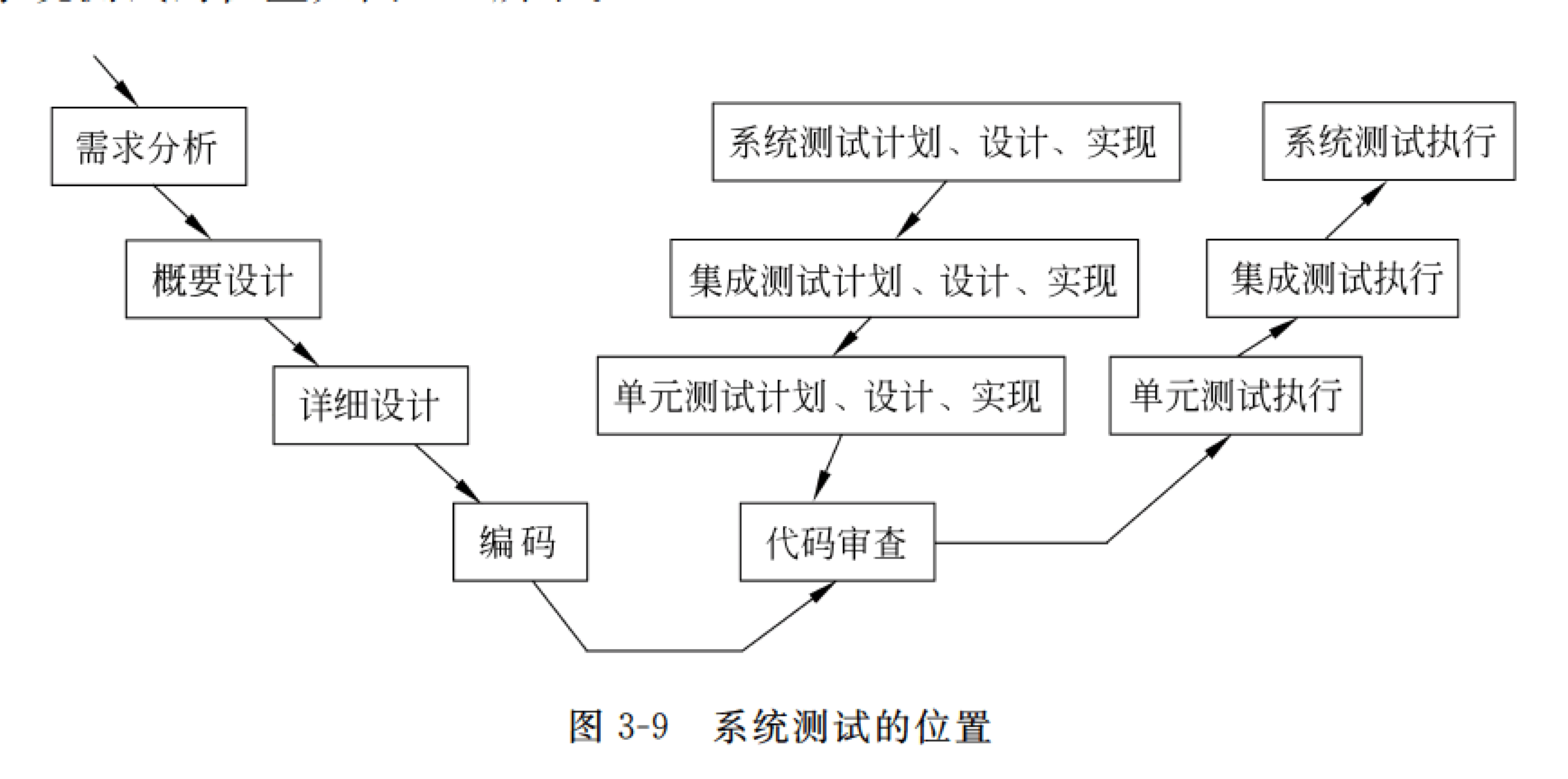


图 3-9　系统测试的位置

3.5　终止测试

3.5.1　终止测试的标准

决定什么时候停止测试是一个非常重要的问题。因为测试不可能找出程序中的所有错误，因此基于费用和时间条件限制，测试最终一定要停止。

黑盒测试和白盒测试都是选择测试，不可能彻底发现程序的所有错误。既然如此，测试何时终止是一个问题。显然，测试过少，程序的遗留错误较多，将降低其可靠性；但过量的测试也会增大软件成本。在此列出了 3 种实用的终止测试标准。

1. 使用特定的测试用例作为判断完成测试的基础

在单元测试时，满足指定的覆盖准则；在功能测试时，由因果图、边界值分析、猜测错误等规范化方法而产生测试用例全部执行作为完成测试的基础。

2. 指出完成测试的要求

如直接指出要查出的错误数，使用该方法的要求如下。

(1) 估计程序中错误的总数。

(2) 估计在这些错误中，通过测试有多少可以很容易地查出来。

(3) 可通过一些方法来估算出程序中的错误总数，估计哪些错误产生于某些特定的设计过程，并且估计这些错误将在测试的哪个阶段被查出。

3. 完成标准

利用经验图标出某个测试阶段中单位时间查出错误的数量和趋势。通过对这一曲线的分析，可以确定应继续测试还是结束测试。

3.5.2 各个测试阶段的终止标准

1. 单元测试停止标准

(1) 单元测试用例设计已经通过评审。
(2) 按照单元测试计划完成了所有规定单元的测试。
(3) 达到了测试计划中关于单元测试所规定的覆盖率的要求。
(4) 被测试的单元每千行代码必须发现至少 3 个错误。
(5) 软件单元功能与设计一致。
(6) 在单元测试中发现的错误已经得到修改，各级缺陷修复率达到标准。

2. 集成测试停止标准

(1) 集成测试用例设计已经通过评审。
(2) 按照集成构件计划及增量集成策略完成了整个系统的集成测试。
(3) 达到了测试计划中关于集成测试所规定的覆盖率的要求。
(4) 被测试的集成工作版本每千行代码必须发现 2 个错误。
(5) 集成工作版本满足设计定义的各项功能、性能要求。
(6) 在集成测试中发现的错误已经得到修改，各级缺陷修复率达到标准。

3. 确认测试停止标准

(1) 确认测试用例设计已经通过评审。
(2) 按照确认测试计划完成了系统测试。
(3) 达到了测试计划中关于确认测试所规定的覆盖率的要求。
(4) 被测试的系统每千行代码必须发现 1 个错误。
(5) 系统满足需求规格说明书的要求。
(6) 在确认测试中发现的错误已经得到修改，各级缺陷修复率达到标准。

4. 系统测试停止标准

(1) 系统测试用例设计已经通过评审。
(2) 按照系统测试计划完成了系统测试。
(3) 达到了测试计划中关于系统测试所规定的覆盖率的要求。
(4) 被测试的系统每千行代码必须发现 1 个错误。
(5) 系统满足各相关部件的要求。
(6) 在系统测试中发现的错误已经得到修改，各级缺陷修复率达到标准。

5. 软件测试停止标准

(1) 软件系统经过单元、集成、确认、系统测试，分别达到单元、集成、确认、系统测试停止标准。

(2) 软件系统通过验收测试,并已得出验收测试结论。

(3) 软件项目需暂停以进行调整时,测试应随之暂停,并备份暂停点数据。

(4) 软件项目在其开发生存周期内出现重大估算、进度偏差,需暂停或终止时,测试应随之暂停或终止,并备份暂停或终止点数据。

小　　结

本章对软件测试的过程进行了详细的说明,对单元测试、集成测试、确认测试和系统测试进行了具体的介绍与描述。通过这些内容的学习,能够理解和掌握软件测试过程,并可以根据不同的项目选择测试用例。最后介绍了各项测试终止的标准,为完成终止测试建立了基础。

习　题　3

1. 单元测试要处理的5个问题是:________,________,________,________,________。

2. 检查软件产品是否符合需求定义的过程称为(　　)。

A. 确认测试　　B. 集成测试　　C. 验收测试　　D. 验证测试

3. 按照集成测试中自顶向下集成的策略,图3-10所示程序结构中各模块是按照什么次序集成的?

4. 按照集成测试中混合集成测试的策略,图3-11所示程序结构中各模块是怎样集成的?

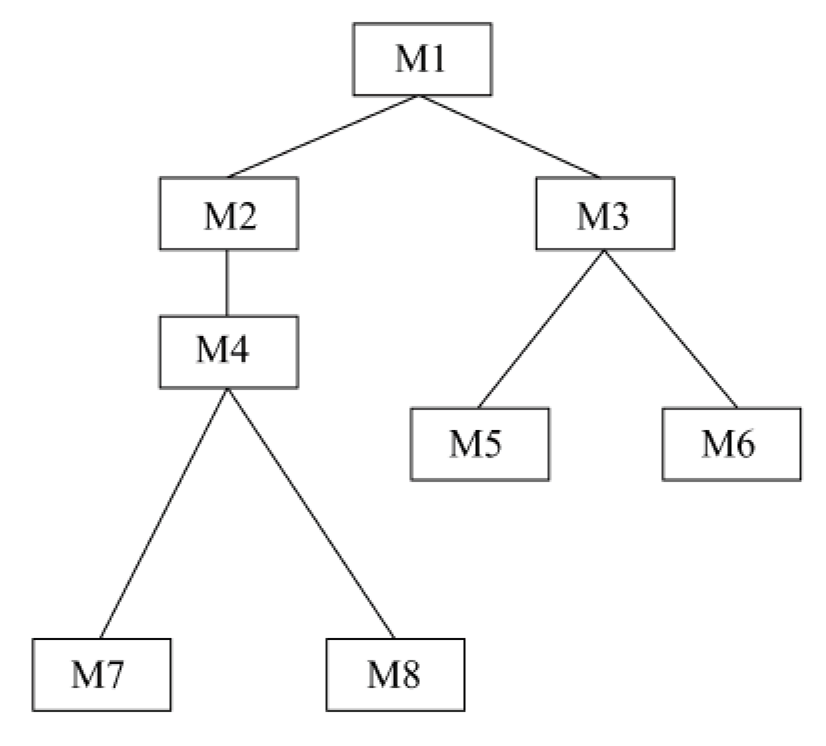


图3-10　自顶向下集成测试策略

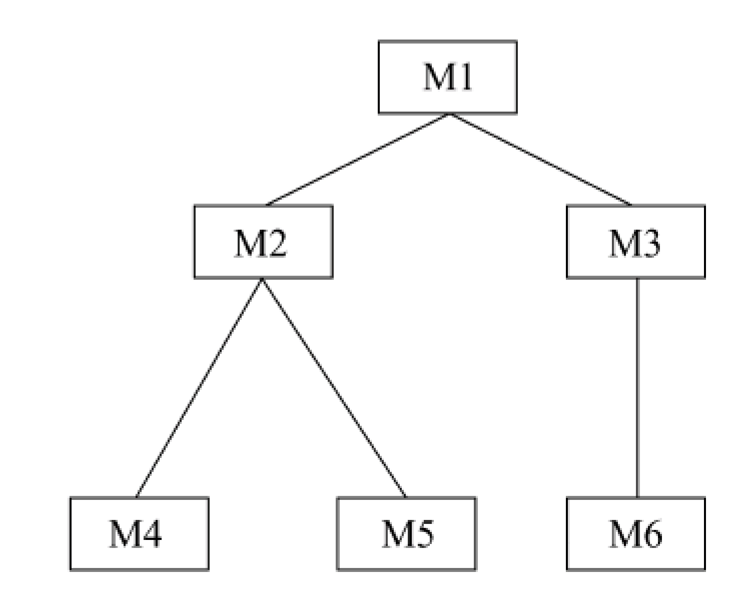


图3-11　混合集成测试策略

5. α测试与β测试的区别。

6. 系统测试与单元测试、集成测试的区别。

第 4 章　面向对象软件测试

学习要点：

- 面向对象测试层次。
- 面向对象测试模型。
- 类测试。
- 面向对象集成测试。
- 面向对象系统测试。

4.1　面向对象测试基础

面向对象软件与结构化软件相比较，存在严重的测试问题，但 UML 语言的出现推动了面向对象技术的发展。在面向对象软件测试中，存在许多值得研究的问题，如面向对象单元划分问题、利用传统测试方法解决面向对象测试问题、类测试、集成测试和系统测试等一系列问题。在本书中，将面向对象软件测试简称面向对象测试。

4.1.1　面向对象测试层次

在面向对象测试中，通常分为 3 个层次，把类看作单元，即类测试、集成测试和系统测试。其中面向对象单元测试主要对类中的成员函数及成员函数间的交互进行测试；面向对象集成测试主要对系统内部的相互服务进行测试，如类间的消息传递等；面向对象系统测试是基于面向对象集成测试的最后阶段的测试，主要以用户需求为测试标准。

4.1.2　面向对象测试顺序

一个类簇由一组相关的类、类树或类簇组成。类的继承关系、组装关系以及类簇包含关系可以构造相应的层次结构，而这些层次结构也就决定了测试顺序。对于继承结构测试次序是父类在先，子类在后。父类可看作其子类的公共部分，在父类测试完成的前提下，子类测试可以关注子类独有的部分以及父类和子类之间的交互。组装结构的测试顺序是部分类在先整体类在其后，在部分对象类测试安全的前提下，整体对象类的测试可以关注各个部分类是否能够按规约进行组装。类簇包含关系测试顺序，先测试组成类簇的各个部件，而后对这些部件进行集成测试。

4.1.3　测试用例

在面向对象的软件中,类需要测试,从而需要构造相应的测试用例。但是,由于类的重用及系统中类定义结构支持测试用例的重用,因而与传统的软件开发方法相比,测试的开销相应地减少。可充分利用基类的测试和继承关系,重用父类的测试用例,并可利用当前类与其祖先的层次关系渐增地开发测试用例,这称之为层次型渐增测试。测试用例的重用使得系统测试者无须对系统中所有的类分别设计测试用例,而可根据类的引用继承关系,充分地引用继承其测试用例。

面向对象测试用例设计的主要原则如下。

(1) 应唯一标识每个测试用例,并且将其与被测试的类显式地关联。

(2) 应该说明测试的目的。

(3) 测试用例内容为:列出所要测试的对象的专门说明;列出将要作为测试结果运行的消息和操作;列出测试对象时可能发生的例外情况;列出外部条件(即为了适当地进行测试而必须存在的外部环境的变化);列出为了帮助理解或实现测试所需要的补充信息。

测试用例的两种用于生成方法是:①重复使用其他类的测试用例;②通过分析所开发的产品来选择新的测试用例。

4.2　面向对象测试模型

传统的结构化软件测试模型采用了功能细化的观点来检测分析和设计的结果,对面向对象软件已不适用。面向对象的开发模型突破了传统的瀑布模型,将开发分为面向对象分析(OOA)、面向对象设计(OOD)和面向对象编程(OOP)共3个阶段。分析阶段产生整个问题空间的抽象描述,在此基础上,进一步归纳出适用于面向对象编程语言的类和类结构,最后形成代码。基于面向对象的特点,采用上述开发模型能有效地将分析设计的文本或图表代码化,不断适应用户需求的变动。针对开发模型,结合传统的测试步骤的划分方法,出现了面向对象软件开发全过程中不断测试的测试模型,使开发阶段的测试与编码完成后的单元测试、集成测试、系统测试成为一个整体,如图4-1所示。

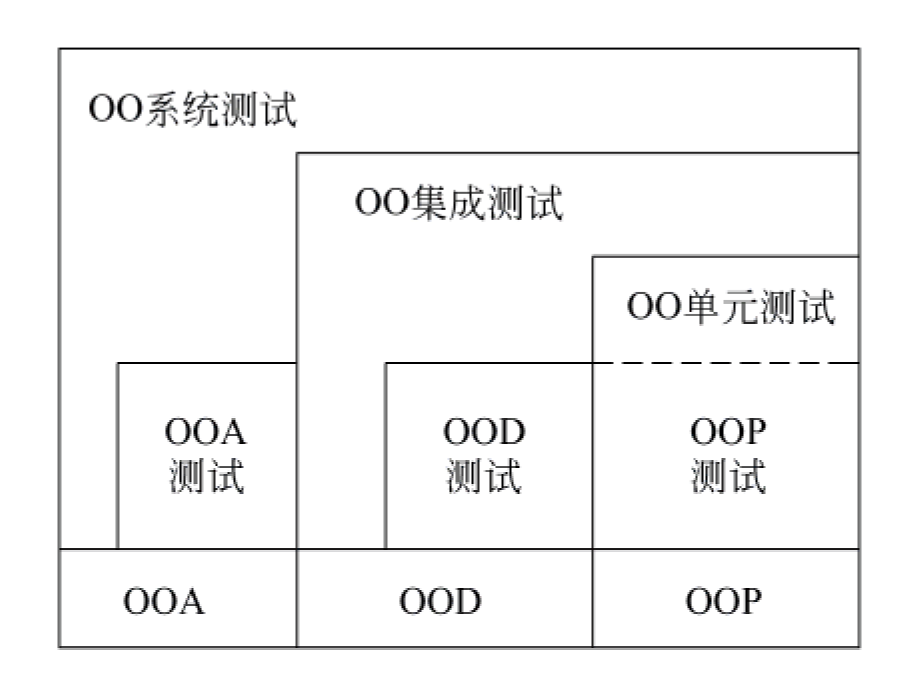


图4-1　OO测试模型

OOA测试和OOD测试是针对OOA结果和OOD结果的测试,主要对分析设计产生的文本进行测试,是软件开发前期的关键性测试。OOP测试是针对编程风格和程序代码实现进行测试,其主要的测试内容在面向对象单元测试和面向对象集成测试中体现。面向对象单元测试是对程序内部具体单一的功能模块的测试,如果程序是用C++语言实现的,主

要就是对类成员函数的测试。面向对象单元测试是进行面向对象集成测试的基础。面向对象集成测试主要对系统内部的相互服务进行测试，例如成员函数间的相互作用、类间的消息传递等。面向对象集成测试不但要基于面向对象单元测试，更要参见 OOD 或 OOD 测试结果。面向对象系统测试是基于面向对象集成测试的最后阶段的测试，主要以用户需求为测试标准，需要借鉴 OOA 或 OOA 测试结果。

4.2.1 面向对象分析的测试

面向过程分析是一个功能分解的过程，把一个系统看成可以分解的功能的集合。这种传统的功能分解分析法的基点是考虑一个系统需要什么样的信息处理方法和过程，通过过程的抽象来处理系统的需求。而 OOA 是把 E-R 图和语义网络模型与面向对象程序语言中的概念相结合而形成的分析方法，最后得到的是以图表形式描述的问题空间。

OOA 全面地将问题空间中要实现的功能抽象化。将问题空间中的实例抽象为对象，用对象的结构反映问题空间的复杂实例和复杂关系，用属性和服务表示实例的特性和行为。对一个系统而言，与传统分析方法产生的结果相反，行为相对稳定，结构相对不稳定，这充分反映了实际问题的特性。OOA 的结果是为后面阶段中类的选定和实现、类层次结构的组织和实现提供平台。因此，OOA 对问题空间分析抽象的不完整性将影响软件的功能实现，导致软件开发后期出现大量原来可避免的修补工作；而一些冗余的对象或结构会影响类的选定、程序的整体结构或增加程序员不必要的工作量。因此，对 OOA 的测试重点是在其完整性和冗余性方面。

OOA 测试分为以下 5 个方面。

- 对象测试。
- 结构测试。
- 主题测试。
- 属性和实例关联的测试。
- 服务和消息关联的测试。

1. 对象测试

在 OOA 测试中，对象是对问题空间中的结构、其他系统、设备、被记忆的事件等实例的抽象。测试考虑如下内容。

(1) 认定的对象是否全面，问题空间中所有涉及的实例是否都反映在认定的抽象对象中。

(2) 认定的对象是否具有多个属性。只有一个属性的对象通常应看成其他对象的属性，而不是抽象为独立的对象。

(3) 对认定为同一对象的实例是否有共同的、区别于其他实例的共同属性。

(4) 对认定为同一对象的实例是否提供或需要相同的服务，如果服务随着不同的实例而变化，认定的对象就需要分解或利用继承性来分类表示。如果系统没有必要始终保持对象代表的实例的信息，提供或者得到关于它的服务，认定的对象也无必要。

(5) 认定的对象的名称应该尽量准确、适用。

2. 结构测试

认定的结构指的是多种对象的组织方式，反映了问题空间中的复杂实例和复杂关系。认定的结构可分为分类结构和组装结构两种。分类结构体现了问题空间中实例的一般与特殊的关系，组装结构体现了问题空间中实例的整体与局部的关系。

1）对认定的分类结构的测试内容

（1）对于结构中的一种对象，尤其是处于高层的对象，是否在问题空间中含有不同于下一层对象的特殊可能性，即是否能派生出下一层对象。

（2）对于结构中的一种对象，尤其是处于同一低层的对象，是否能抽象出在现实中有意义的更一般的上层对象。

（3）对所有认定的对象，是否能在问题空间内向上层抽象出在现实中有意义的对象。

（4）高层的对象的特性是否完全体现下层的共性。

（5）低层的对象是否有高层特性基础上的特殊性。

2）对认定的组装结构的测试内容

（1）整体（对象）和部件（对象）的组装关系是否符合现实的关系。

（2）整体（对象）和部件（对象）是否在考虑的问题空间中有实际应用。

（3）整体（对象）中是否遗漏了反映在问题空间中有用的部件（对象）。

（4）部件（对象）是否能够在问题空间中组装新的有现实意义的整体（对象）。

3. 主题测试

主题是在对象和结构的基础上更高一层的抽象，为了提供 OOA 分析结果的可见性，如同文章的各部分内容的概要。对主题层的测试应该考虑以下方面。

（1）如果主题个数超过 7 个，就要求对有较密切属性和服务的主题进行归并。

（2）主题所反映的一组对象和结构是否具有相同和相近的属性和服务。

（3）认定的主题是否是对象和结构更高层的抽象，是否便于理解 OOA 结果的概貌（尤其是对非技术员是否易于理解 OOA 的结果）。

（4）主题间的消息联系（抽象）是否代表了主题所反映的对象和结构之间的所有关联。

4. 属性和实例关联的测试

属性是用来描述对象或结构所反映的实例的特性。而实例关联是反映实例集合间的映射关系。对属性和实例关联的测试考虑如下。

（1）定义的属性是否对相应的对象和分类结构的每个现实实例都适用。

（2）定义的属性在现实世界是否与这种实例关系密切。

（3）定义的属性在问题空间是否与这种实例关系密切。

（4）定义的属性是否能够不依赖于其他属性被独立理解。

（5）定义的属性在分类结构中的位置是否恰当，低层对象的共有属性是否在上层对象属性中体现。

（6）在问题空间中每个对象的属性是否定义完整。

（7）定义的实例关联是否符合现实。

(8) 在问题空间中实例关联是否定义完整,特别需要考虑一对多和多对多的实例关联。

5. 服务和消息关联的测试

定义服务就是定义的每一种对象和结构在问题空间中所要求的行为。由于问题空间与实例间存在必要的通信,在 OOA 中相应地需要定义消息关联。对定义的服务和消息关联的测试从如下方面进行。

(1) 对象和结构在问题空间的不同状态是否定义了相应的服务。

(2) 对象或结构所需要的服务是否都定义了相应的消息关联。

(3) 定义的消息关联所指引的服务提供是否正确。

(4) 沿着消息关联执行的线程是否合理,是否符合现实过程。

(5) 定义的服务是否重复,是否定义了能够得到的服务。

4.2.2 面向对象设计的测试

结构化设计方法是把对问题域的分析转化为对求解域的设计,分析的结果是设计阶段的输入。面向对象设计以面向对象分析为基础归纳出类,并建立类结构和构造类库,实现分析结果对问题空间的抽象。OOD 归纳的类是各个对象的相同或相似的服务。由此可见,由 OOD 确定的类和类结构不仅能满足当前需求分析的要求,更重要的是通过重新组合或加以适当的补充,能方便实现功能的重用和扩增,以不断适应用户的要求。因此,OOD 的测试是对功能的实现和重用以及对 OOA 结果的拓展,主要内容包括以下几个。

- 对认定的类的测试。
- 对构造的类层次结构的测试。
- 对类库支持的测试。

1. 对认定的类的测试

OOD 认定的类可以是 OOA 中认定的对象,也可以是对象所需要的服务的抽象,对象所具有的属性的抽象。认定的类应尽量基础化,以便有利于维护和重用,需要测试认定的类,主要内容如下。

(1) 是否涵盖了 OOA 中所有认定的对象。

(2) 是否能体现 OOA 中定义的属性。

(3) 是否能实现 OOA 中定义的服务。

(4) 是否对应着一个含义明确的数据抽象。

(5) 是否尽可能少地依赖其他类。

(6) 类中的方法是否单用途。

2. 对构造的类层次结构的测试

为能充分发挥面向对象的继承共享特性,OOD 的类层次结构通常是基于从 OOA 中产生的分类结构的原则来组织的,着重体现父类和子类间的一般性和特殊性。在当前的问题空间,对类层次结构的主要要求是能在解空间中构造实现全部功能的结构框架。为此,测试下述内容。

(1) 类层次结构是否涵盖了所有定义的类。

(2) 是否能体现 OOA 中所定义的实例关联。

(3) 是否能实现 OOA 中所定义的消息关联。

(4) 子类是否具有父类没有的新特性。

(5) 子类间的共同特性是否完全在父类中得以体现。

3. 对类库支持的测试

虽然对类库的支持属于类层次结构的问题,但强调的是软件开发的重用。由于它并不直接影响当前软件的开发和功能实现,因此,可以将其单独提出来测试,也可作为对高质量类层次结构的评估。测试要点如下。

(1) 在一组子类中,关于某种含义相同或基本相同的操作是否有相同的接口(包括名字和参数表)。

(2) 类中的方法功能是否较单纯,相应的代码行是否较少(建议不超过 30 行)。

(3) 类的层次结构是否是深度大,宽度小。

4.2.3 面向对象编程的测试

面向对象程序具有继承、封装和多态的特性。封装是对数据的隐藏,外界只能通过操作来访问或修改数据,降低了数据被任意修改和读写的可能性,降低了传统程序中对数据非法操作的测试。继承是面向对象程序的重要特点,继承提高了代码的重用率。多态使得面向对象程序呈现出强大的处理能力,但同时使得程序内同一函数的行为复杂化,测试时必须考虑不同类型代码和产生的行为。

面向对象程序把功能的实现分布在类中。类通过消息传递来协同实现设计要求的功能。正是这种面向对象程序风格,将出现的错误能精确地确定在某一具体的类上。因此,在面向对象编程阶段要忽略类功能实现的细则,将测试集中在类功能的实现和相应的面向对象程序风格上面,主要体现为以下两个方面(以C++ 语言为例)。

1. 数据成员要满足数据封装的要求

数据封装是数据及对数据操作的集合。检查数据成员是否满足数据封装的要求,基本原则是数据成员是否被外界(数据成员所属的类或子类以外的调用)直接调用。更直观地说,当改变数据成员的结构时,是否影响了类的对外接口,是否会导致相应外界必须改动。有时强制的类型转换会破坏数据的封装特性。

例如:

```
class Hiden
{
private:
    int a=1;
    char * p="hiden";
};
class Visible
```

```
{
public:
    int b=2;
    char * s="visible";
};
⋮
Hiden pp;
Visible * qq=(Visible*)&pp;
```

在上面的程序段中,通过 qq 可随意访问 pp 的数据成员。

2. 类实现了要求的功能

类功能的实现是通过类的成员函数执行的。在测试类的功能实现时,首先保证类成员函数的正确性。单独地看待类的成员函数,与面向过程程序中的函数或过程没有本质的区别,几乎所有传统的单元测试中所使用的方法,都可在面向对象的单元测试中使用。类函数成员的行为是类功能实现的基础,类成员函数间的作用和类之间的服务调用是单元测试无法确定的。因此,需要进行面向对象的集成测试。需要声明测试类的功能,不能仅满足于代码能无错运行或被测试的类所提供的功能无错,应该以 OOD 结果为依据,检测类提供的功能是否满足设计的要求,是否有缺陷。如果通过 OOD 结果仍有模糊的地方,应以 OOA 的结果为最终标准。

4.3 类 测 试

传统的过程软件测试技术形成了较完整的理论体系,其中测试方法、测试用例的产生方法等都趋于完善。然而,面向对象软件开发方法的出现,使软件测试员面临新的课题。一方面,面向对象技术产生了更好的软件体系结构、更规范的编程风格,极大优化了数据使用的安全性,提高了代码的重用性和程序的可维护性,正逐步地取代广泛使用的面向过程的开发方法,是解决软件危机的新技术;另一方面,由于面向对象技术所特有的多态性、继承性、封装性、动态链接及类间的松散耦合等特点,产生了传统设计语言不可能出现的错误,给软件测试提出了新的要求。面向对象测试的目标与过程软件的测试目标相同,即使用最小的工作量发现最多的错误。但是面向对象测试的策略和技术与传统测试不同。由于面向对象程序本身所具有的特性,使得测试的范围扩大到复审分析和设计模型。测试的关键点是类,类是面向对象方法中最重要的内容,是面向对象程序的基本单位。而传统的过程测试的方法只适用于类中方法的测试,不适用于类的整体测试,同时仅检查类中方法的正确性不能保证类在整体上是正确的。类测试由验证类的实现是否和该类的说明完全一致的相关活动组成,如果类的实现正确,那么类的每个实例的行为也应该正确。

4.3.1 类测试的概述

1. 类测试的概念

类测试由类和测试体构成,测试时通过运行测试体来验证类的实现和类的描述是否

一致，如果类的实现正确，那么表示该类的所有实例行为也正确。因此，被测试的类必须有正确而且完整的描述，也就是说，测试的类在设计阶段产生的所有要素正确而且完整。

2. 类测试顺序

类是构成面向对象软件的基本单元，类测试是面向对象软件测试的关键。类测试时不仅要将操作作为类的一部分，同时要把对象与其状态结合起来，进行对象状态行为的测试。类的测试按顺序分为以下 3 个部分。

(1) 基于服务的测试：测试类中的每一个方法。

(2) 基于状态的测试：考察类的实例在其生存周期各个状态下的情况。

(3) 基于响应状态的测试：从类和对象的责任出发，以外界向对象发送特定的消息序列的方法来测试对象的各个响应状态。

类测试的示意图如图 4-2 所示。

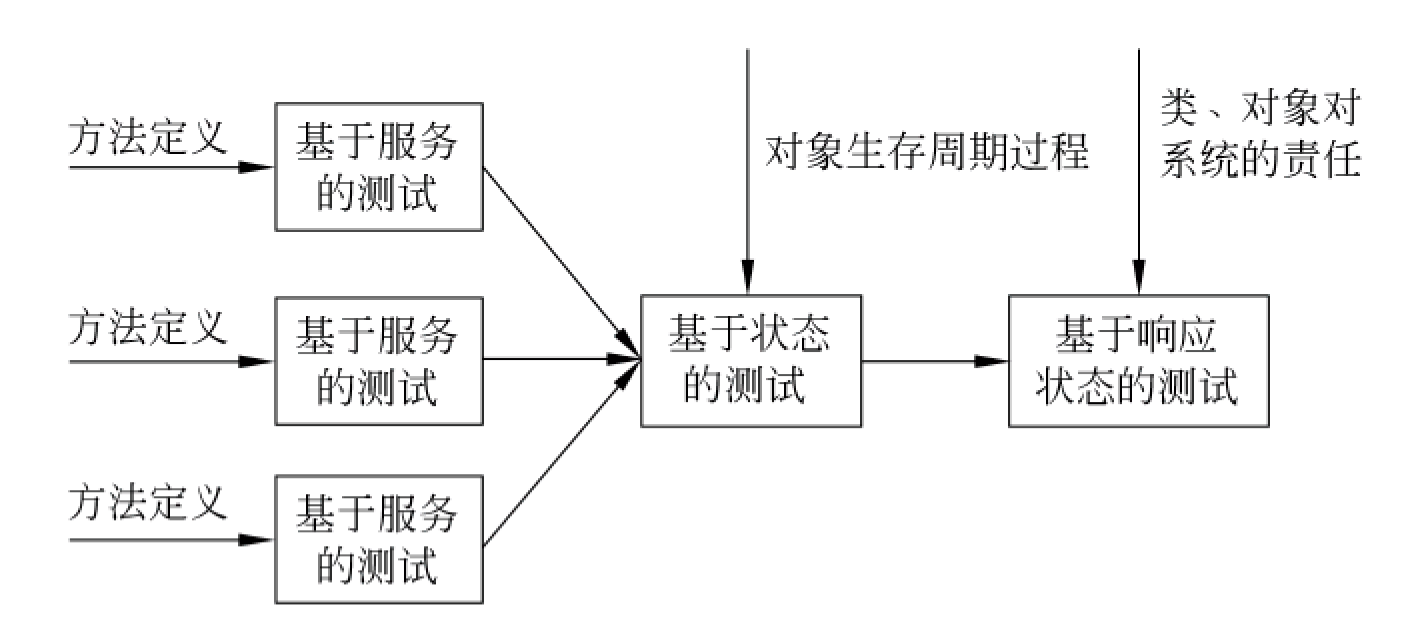


图 4-2　类测试

3. 类测试组成

将为了实现类测试而构造的可执行程序，分为测试用例和测试驱动两部分。测试用例是静态的，包含了一个或多个需验证的单元；测试驱动是动态的，可将一个或多个测试用例运行起来。

一个测试用例可以由 1～n 个测试描述组合而成，一个测试用例又可以由 1～n 个测试用例聚合而成，这就是测试组成的框架。

在类测试时先确定类的测试用例，然后加载类的测试驱动来运行这些测试用例，同时给出运行测试用例后的结果。

在进行类测试时，要考虑测试人员、测试内容、测试时间、测试程度和测试过程等内容。

4. 类测试的过程

(1) 用测试驱动来驱动该类的测试用例，根据测试用例的指定情况生成用例所需要的测试环境。

(2) 在这个测试环境中，测试驱动向被测实例发送一个或多个需要验证的消息。

(3) 然后根据被测实例的状态变化、响应值、返回参数和结果，来判断验证消息是否

通过。

(4) 在所有测试用例验证执行后,释放所有已生成的实例和值域。

如果类包含了静态属性或者静态方法,同样需要对其进行测试。这些静态属性和方法属于类本身,表明该类本身就是一个对象,无须实例化就可以直接调用这些属性和方法。

5. 类测试程度

类的测试程度取决于测试了多少类实现和类描述,由于每个类被实例化后都有自己的状态,状态的不同影响着操作的具体含义,对于任何一个类进行测试都需要进行综合考虑,所以不可能采取穷举的方法来进行测试,从而使对不同类的测试程度估算比较困难。例如接口类中包含了一组方法,有个具体类实现了它,在具体类测试用例中加入了对该接口方法的测试,如果该具体类的其他实例方法没有和任何类交互,那么该类的测试覆盖率就是百分之百的。可以采取组合测试或者在多个测试用例中抽取重要的、覆盖最大的测试用例来进行测试。

6. 类测试员

类测试员可由类设计员和类实现员组成,类设计员在设计类的过程中了解了类的所有实现细节,从而可以设计出更具有针对性和高覆盖率的类测试用例,类设计员对类的说明要准确,类设计员可以通过测试复审排除任何误解或者在设计中引入同行评审制度,通过互相检查来避免测试缺陷发生。具体的测试活动一般由类实现员操作,因为类实现员极其熟悉该类代码,但是如果类测试都是由独立的测试员来做,那么就需要类测试员有很好的类描述技巧。

7. 测试时间

类测试可以在软件开发过程的不同位置进行,但在目前,各种软件开发模型的显著特点是循环递增,这种方式决定了一个类的描述和实现可能会发生变化,所以应该在软件的其他部分使用被测类之前对该类进行测试。在测试过程中每当类的实现发生变化时,需要采用回归测试,也就是说,类测试应该与类的设计、开发保持同步关系。

8. 类测试的充分性

穷举测试一般不可能实现,但如果不使用穷举测试就不能保证一个类的每一方面都符合它的说明。针对震中问题,可以采用折中的方法,即利用充分性的标准来衡量测试结果。充分性的标准如下所述。

1) 基于状态的覆盖率

基于状态的覆盖率是指测试覆盖了多少状态转换作为类测试充分性的根据。如果测试没有覆盖状态转换,表明测试不充分。但是即使测试用例对所有状态都覆盖了一次,由于状态通常包含了各种对象属性的值域,测试的充分性应不能保证。

2) 基于约束的覆盖率

基于约束的覆盖率是指根据有多少对前置条件和后置条件被覆盖来表示充分性。如果测试用例包含了所有前置条件和后置条件的组合情况,那么就符合充分性标准。

3）基于代码的覆盖率

基于代码的覆盖率是指当所有的测试用例都执行结束时，确定实现一个类的每一行代码，或代码通过的每一条路径至少执行一次。

在上述标准中，如果基于代码的覆盖率达到了100%，但也不能保证基于约束的覆盖率和基于状态的覆盖率达到100%，所以，对于具体问题，充分性度量标准的选择异常重要。

9. 类测试与单元测试的比较

面向对象软件从宏观上来看是各个类之间的相互作用。在面向对象系统中，系统的基本构造模块是封装了的数据和方法的类和对象，而不再是一个个能完成特定功能的功能模块。每个对象有自己的生存周期，有自己的状态。消息是对象之间相互请求或协作的途径，是外界使用对象方法及获取对象状态的唯一方式。对象的功能是在消息的触发下，由对象所属类中定义的方法与相关对象的合作共同完成的，且在不同状态下对消息的响应可能完全不同。工作过程中对象的状态可能被改变，产生新的状态。对象中的数据和方法是一个有机的整体，测试过程中不能仅仅检查输入数据产生的输出结果是否与预期的一致，还要考虑对象的状态。模块测试的概念已不适用于对象的测试。类测试将是整个测试过程的一个重要步骤，它与传统测试方法的区别如图4-3所示。

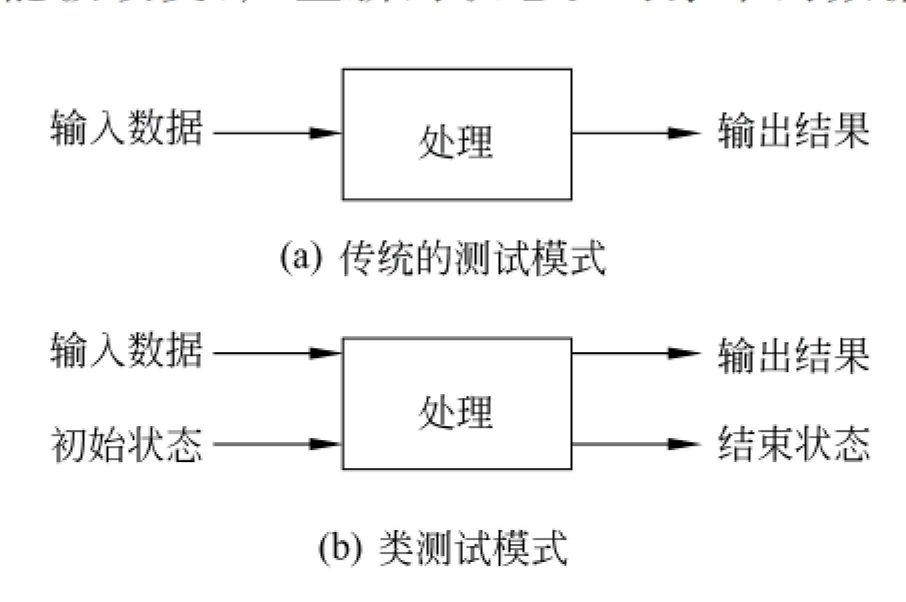


图4-3 两种不同的测试模型

面向对象软件的类测试与传统软件的单元测试相对应，却不是完全一样的。传统单元测试注重单元之间的接口测试，每个单元都有自己的输入输出接口，如果在调用中出现了严重错误，是由单元之间接口的实现引起的，与单元本身没有什么直接关系。而在面向对象中，每一个类就是一组最小的交互单元，其内部封装了各种属性和消息传递方式。类被实例化后产生了对象，每个对象都有相对的生存周期和活动范围，在这个范围内，对象和对象本身、对象和对象之间都可以通过已定义的消息传递方式进行交互。所以，一个对象有它自己的状态和依赖于状态的行为，对象操作既与对象的状态有关，但也可能改变对象的状态。因此，类测试时不仅要将操作作为类的一部分，同时要把对象与其状态结合起来，进行对象状态行为的测试。也就是说，类测试除了需要测试类中包含的类方法，还需要测试类的状态，这是传统单元所没有的。

4.3.2 类测试技术

类测试可以使用多种方法，例如随机测试，划分测试，基于故障、规约、流图、状态的类测试技术等，在这里仅介绍基于服务的和基于状态的两种类测试的方法。

1. 基于服务的类测试技术

基于服务的类测试主要检测封装在类中的方法对数据进行的操作，可以采用传统

的白盒测试方法，如控制流测试、数据流测试、循环测试、排错测试、分域测试等。尽管人们对类的测试内容的认识比较一致，但由于受面向对象软件测试技术发展水平的限制，测试员在选择测试用例时往往都是根据直觉和经验来进行的，给测试带来很大的随机性，并且由于测试员的个性及局限性也使得选择的测试用例仅能测试出其所熟悉的某一方面的错误，许多隐含的其他错误不能被检测出来，降低了软件的可靠性。为了克服软件测试的随机性和局限性，保证测试的质量，提高软件的可靠性，有必要寻找好的测试模型。

1）基于服务的类测试模型

块分支图(Block Branch Diagram，BBD)是一种比较好的基于类服务的测试模型。服务 f 的 BBD 是一个五元组，如图 4-4 所示。

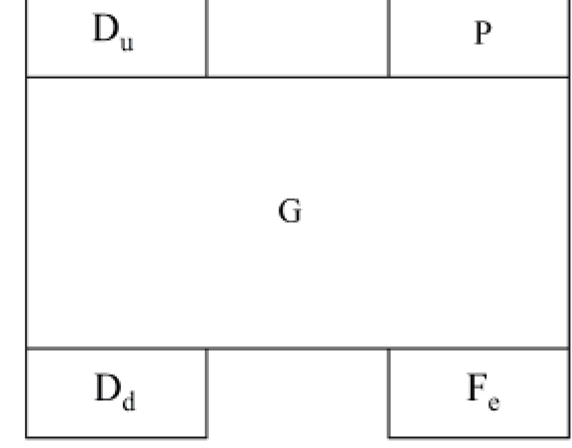


图 4-4 服务 f 的 BBD 五元组

其中：BBD $f=(D_u,D_d,P,F_e,G)$；

$D_u=\{d_i|d_i$ 是 f 引用的全局数据或类数据$\}$；

$D_d=\{d_i|d_i$ 是修改了的全局数据或类数据$\}$；

$P=X_1\theta_1，X_2\theta_2，\cdots，X_n\theta_n；X_{n+1}\theta_{n+1}$ 是 f 的参数表和函数返回值，θ_i 为 ↓（表示输入）、↑（表示输出）或 ↓（表示输入/输出）。若 X_{n+1} 省略，则无返回值；


$F_e=\{f_i|f_i$ 是被 f 调用的其他服务$\}$；

G 是一个有向图，即块体，是按照控制流图的思想修改 f 的程序流图而来的，表示了 f 的控制结构。f 中的复合条件判断被分解，每个判断框只有单个的条件。

【例 1】 对于类 matrix 的服务 addIf，其 BBD 如图 4-5 所示。

matrix∷addIf(…)的源程序如下：

```
Class matrix
{
public:
    int row,col;
    Unsigned addIf(int row1,col1,row2,col2)
    {
        if((row1==row2)&&(col1==col2))
        {
            row=row1;
            col=col1;
            return 1;
        }
        else
            return 0;
    }
};
```

【例 2】 对于类 person 的服务 loadIfWaiting，其 BBD 如图 4-6 所示。源程序如下：

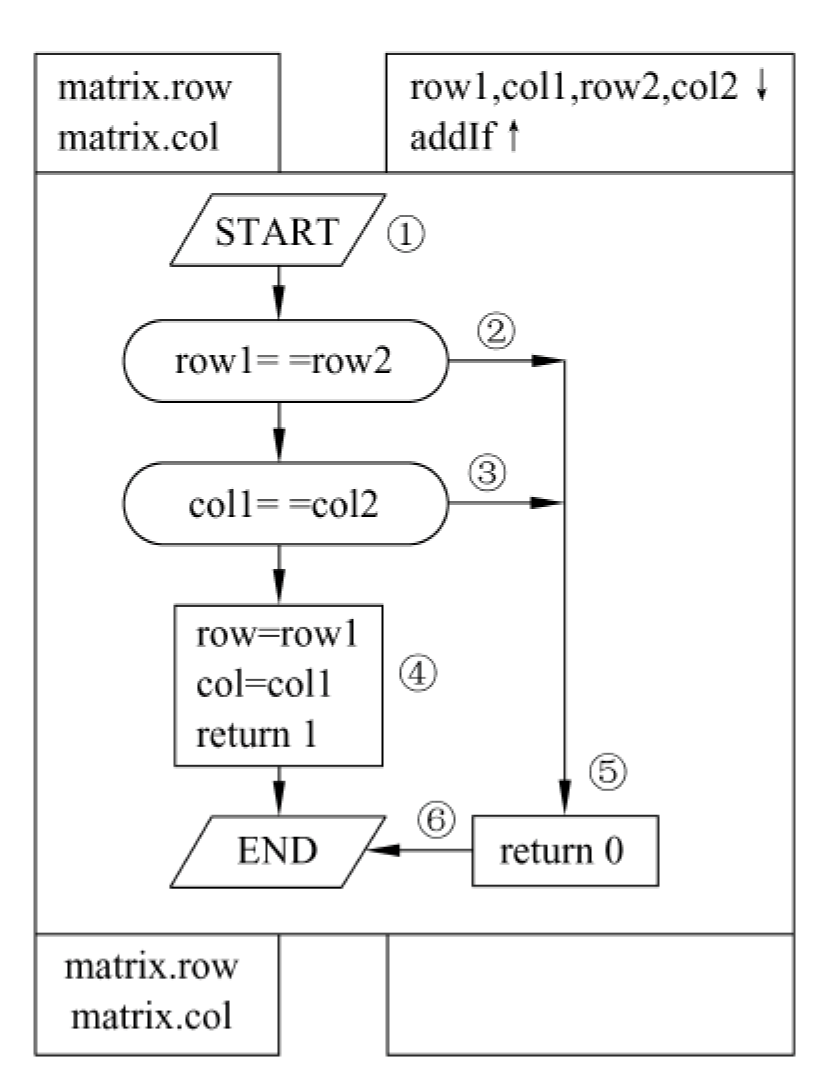


图 4-5　matrix∷addIf(…)的 BBD 图

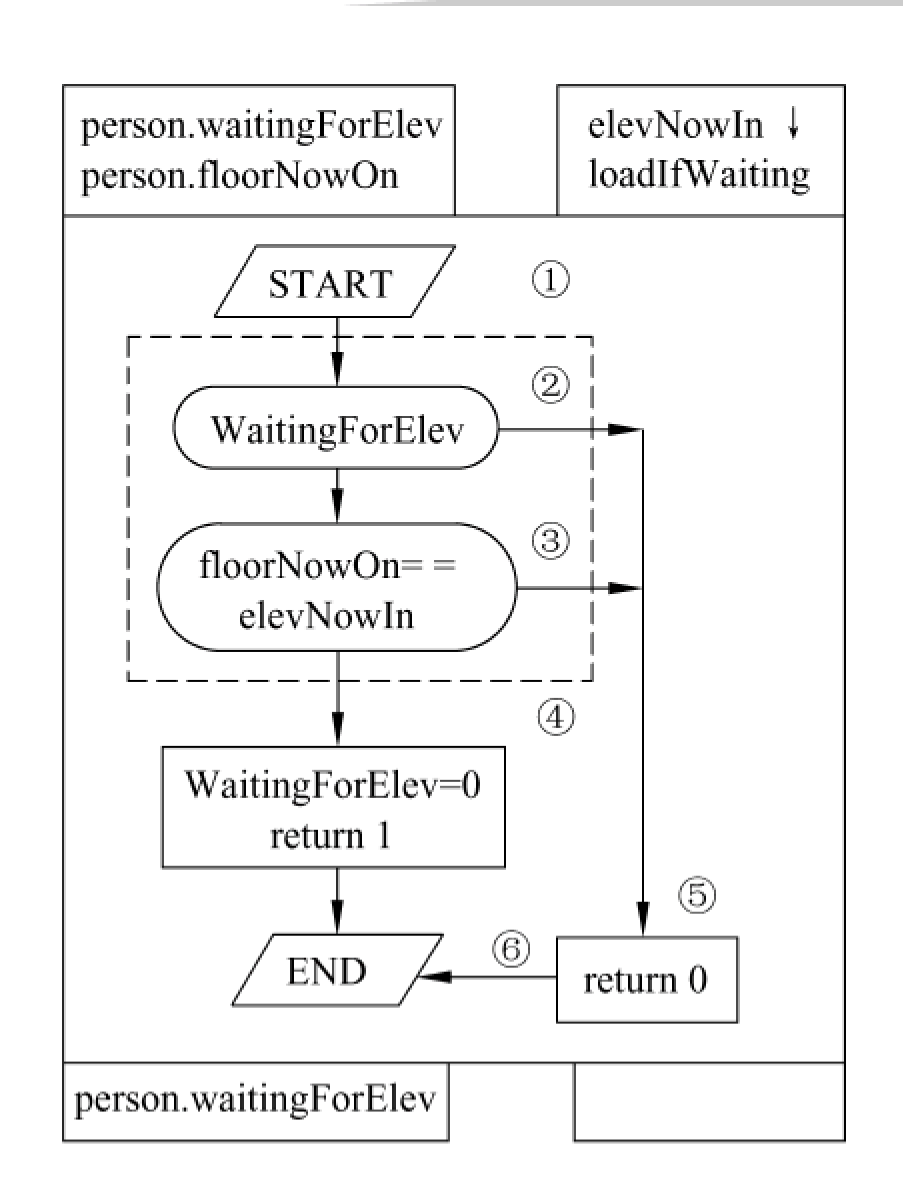


图 4-6　person∷loadIfWaiting(…)的 BBD 图

```
Class person
{
public:
    int waitingForElev, floorNowOn;
    Unsigned loadIfWaiting(int elevNowIn)
    {
        If(waitingForElev &&(floorNowOn= =elevNowIn))
        {                //有人在等待电梯并且他与电梯在同一楼层
            waitingForElve= 0;
            return 1;
        }
        else return 0;
    }
};
```

BBD 通常有两种获取途径。一是采用逆向工程的方法根据源程序画出流程图，然后构造出 BBD。但这毕竟是在缺少软件开发前期的分析、设计文档不齐全的情况下退而求其次的办法，当源程序不正确时构造出来的 BBD 就是错误的。另一种途径就是追根溯源，在软件的分析、设计阶段就根据测试的需要构造出相应的 BBD。这样就能从根本上解决问题，正确地指导类的服务的测试。

借助 BBD，可以对服务进行结构测试和黑盒测试。前者主要进行基本路径测试，此项测试包含了语句覆盖测试和分支覆盖测试。

2）基于服务的类测试策略

（1）白盒测试。白盒测试主要进行基本路径测试。路径测试是一种理想的测试方

法。但是在实际问题中因为路径的组合爆炸使得该方法不可能真正实现。语句覆盖、分支覆盖测试则又太弱,只能检查出有限的错误。基本路径测试正好介于它们之间,此项测试包含了语句覆盖测试和分支覆盖测试,它既可以检查程序主要的执行路径,又可以覆盖所有的分支,而且还能够满足语句覆盖的要求,这是一种较为实用的测试方法。因此,白盒测试主要进行基本路径测试。

基本路径测试是根据软件过程性描述(详细设计或代码),在程序控制流图的基础上确定复杂性度量,通过分析控制结构的环路复杂度,导出基本可执行路径集合,从而设计出测试用例的方法。通过基本路径测试设计出的测试用例要保证在测试中程序的每一个可执行语句至少执行一次。其具体实施步骤如下。

① 绘制服务的控制流图。只要把 BBD 中块体部分的起始节点、结束节点和每个矩形框、判断框分别用它们旁边的数字标记代替,就可以得到该服务的控制流图,如图 4-7 所示。

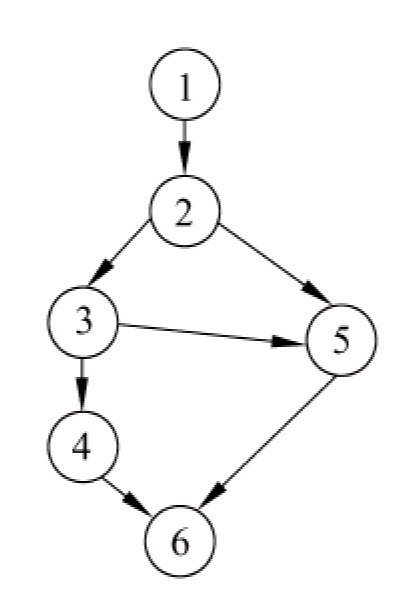


图 4-7 类 person 的服务 loadIfWaiting 的控制流图

② 计算程序环路复杂度。利用控制流图,可以计算该服务的 McCabe 环路复杂度 V(G)。实质上它等于程序中独立路径的条数。一条独立路径是指至少引入一个新处理语句或一个新判断的程序通路。它给出了程序中每个语句至少执行一次所必须进行测试的最少次数。V(G)=E−N+2,其中:E 是流图中边的数量,N 是流图节点数量。由图 4-7 可知,服务 loadIfWaiting 的控制流图有 3 个区域,故其 McCabe 环路复杂度 V(G)=3。

③ 确定基本路径集。可以从环路复杂度确定程序基本路径集中的独立路径条数,这是确保程序中每个可执行语句至少执行一次所必需的测试用例数目的上界。在图 4-7 所示的控制流图中,一组独立的路径如下。

path 1:1-2-3-4-6

path 2:1-2-5-6

path 3:1-2-3-5-6

④ 生成测试用例。根据判断节点给出的条件,选择适当的数据以保证每一条基本路径都可被测试。只要设计出的测试用例能够保证这些路径的执行,就可以使得程序中的每个可执行语句至少执行一次,每个条件的取真和取假分支也能得到测试。满足上述基本路径测试用例如表 4-1 所示。

表中的限制条件是根据 BBD 的块体部分中的简单判断条件得到的。当为某一条路径准备测试数据时,测试数据值必须满足这些限制条件。另外,输入数据部分给出了该服务的输入参数和引用的全局数据和类数据的具体值,以强制特定路径的执行。对于块体中的某些关键路径,如果外部输入不能控制其走向,则应根据设计函数的思想,将局部数据纳入 D_u 中,并且在输入数据部分给出相应的值。

表 4-1　person 程序的测试路径表

<table>
<tr><td colspan="2">服务名称</td><td colspan="3">person∷loadIfWaiting(…)</td></tr>
<tr><td colspan="2">基本路径集</td><td colspan="3">Path:1.2.3.4.6
Path:1.2.5.6
Path:1.2.3.5.6</td></tr>
<tr><td rowspan="2">输入</td><td colspan="2">输入参数</td><td colspan="2">引用数据</td></tr>
<tr><td colspan="2">P={elevNowIn}</td><td colspan="2">Du={waitingForElev,floorNowOn}</td></tr>
<tr><td rowspan="2">输出</td><td colspan="2">输出参数</td><td colspan="2">修改数据</td></tr>
<tr><td colspan="2">P={loadIfWaiting}</td><td colspan="2">Dd={waitingForElev}</td></tr>
<tr><td>路径名称</td><td>限制条件</td><td colspan="2">输入数据</td><td>期望输出</td></tr>
<tr><td>Path1</td><td>waitingForElev=1
floorNowOn==elevNowIn</td><td colspan="2">waitingForElev=1
floorNowOn=1
elevNowIn=1</td><td>loadIfWaiting=1
waitingForElev=0</td></tr>
<tr><td>Path2</td><td>waitingForElev=0</td><td colspan="2">waitingForElev=0
floorNowOn=1
elevNowIn=1</td><td>loadIfWaiting=0
waitingForElev=0</td></tr>
<tr><td>Path3</td><td>waitingForElev=1
floorNowOn≠elevNowIn</td><td colspan="2">waitingForElev=1
floorNowOn=1
elevNowIn=2</td><td>loadIfWaiting=0
waitingForElev=1</td></tr>
</table>

（2）黑盒测试。基本路径测试方法可以检查出源程序中的大部分错误，但是对于一些程序的书写错误，比如在编写源程序的时候错把"<"写成"≤"，基本路径测试方法就很难发现。黑盒测试方法则可以有效地检查出这一类错误。在前面知道黑盒测试中用到的技术为等价类划分和边界值分析技术等。等价划分是一种黑盒测试方法，其主要思想是把程序的输入数据集合按输入条件划分为若干等价类，每一等价类相对于输入条件表示为一组有效或无效的输入，然后为每一等价类设计一个测试用例。这样既可以大大减少测试的次数，又不丢失发现错误的机会。边界值分析技术旨在选择测试用例，强迫程序在边界值上执行。

因此，在进行黑盒测试时，对于选择设计测试用例（包括输入参数和引用数据），应指定以下值。

① 典型值，即数据类型中较为典型的值，或被该服务用来检查某些条件出现的值。

② 边界值，即位于数据类型的边界上的值，是比某一典型值稍大或稍小的值。

③ 其他值，即测试员认为比较重要的其他的值。

在设计测试用例时，根据等价类划分的原理，为每一个测试用例从各自取值集合中选择一个值，并排除掉不可能的组合，然后为每个测试用例指定期望输出。例如，类 person 的服务 loadIfWaiting(…)的黑盒测试用例如表 4-2 所示。

表 4-2　类的服务的功能测试用例

<table>
<tr><td colspan="2">服务名称</td><td colspan="2">person::loadIfWaiting(…)</td></tr>
<tr><td rowspan="2">输入</td><td colspan="2">输入参数</td><td>引用数据</td></tr>
<tr><td colspan="2">P={elevNowIn}</td><td>Du={waitingForElev,floorNowOn}</td></tr>
<tr><td rowspan="2">输出</td><td colspan="2">输出参数</td><td>修改数据</td></tr>
<tr><td colspan="2">P={loadIfWaiting}</td><td>Dd={waitingForElev}</td></tr>
<tr><td>输入数据名称</td><td>典 型 值</td><td>边 界 值</td><td>其 他 值</td></tr>
<tr><td>elevNowIn</td><td>{0,1,−1,MAXELEVS}</td><td>{MAX int,MIN int}</td><td>{−1, −2, 2, MAXLEVS, MAXELEVS−1}</td></tr>
<tr><td>waitingForElev</td><td>{0,1,−1}</td><td>{MAX int,MIN int}</td><td>{−2,2}</td></tr>
<tr><td>floorNowOn</td><td>{0,1,−1,MAXELEVS}</td><td>{MAX int,MIN int}</td><td>{−1, −2, 2, MAXLEVS, MAXELEVS−1}</td></tr>
<tr><td>测试用例序号</td><td colspan="2">输 入 数 据</td><td>期 望 输 出</td></tr>
<tr><td>1</td><td colspan="2">waitingForElev=0
elevNowIn=1
floorNowOn=0</td><td>loadIfWaiting=0
waitingForElev=0</td></tr>
<tr><td>2</td><td colspan="2">waitingForElev=1
floorNowOn=MAXELEVS
elevNowIn=MAXELEVS</td><td>loadIfWaiting=1
waitingForElev=0</td></tr>
</table>

黑盒测试中的等价类划分和边界值分析是行之有效的。类的服务的测试模型较好地支持了基本路径测试和等价类划分、边界值分析的测试方法，可以帮助测试员比较全面地、有针对性地构造测试用例，从而有效地克服了软件测试的盲目性和局限性，保证了测试的质量，提高了软件的可靠性。

2. 基于状态的类测试

1) 基于状态的类测试概念

类测试主要考察封装在类中的方法和属性的相互作用。对象具有自己的状态，对对象的操作很可能改变对象的状态，因此，类测试时要把对象与其状态结合起来进行对象状态行为的测试。

基于状态的测试是通过检查对象的状态在执行某个方法后是否会转移到预期状态的一种测试技术，它是面向对象软件测试的重要部分，同传统的控制流和数据流测试相比，它侧重于对象的动态行为，这种动态行为依赖于对象的状态。测试对象动态行为能检测出对象成员函数之间通过对象状态进行交互时产生的错误。因为对象的状态是通过对象的数据成员的值反映出来的，所以检查对象的状态实际上就是跟踪监视对象数据成员的值的变化。如果某个方法执行后对象的状态未能按预期的方式改变，则说明该方法含有错误。

2) 基于状态的类测试内容

状态测试包括类实例化测试和对象状态的测试。

（1）类是属性和方法的封装体，是一个抽象定义的概念，只有经过实例化才能使用。对象是类的实例化，这一过程是通过类的构造函数和析构函数来完成的。类的实例化测试要完成的就是对构造函数和析构函数的测试。

（2）基于对象状态的测试是考察类的实例在生存周期各个状态下的情况。完成类的实例化后，开始对对象可能的状态进行测试，即对象在其生存周期各个状态下的情况，以外界向对象发送特定消息序列的方法来测试对象的响应状态。因为对象的状态是通过对象的数据成员的值反映出来的，所以检查对象的状态实际上就是跟踪监视对象数据成员的值的变化。进行对象状态测试所需的环境，可以通过预置条件和预处理过程实现，这是本阶段的难点。

3）基于状态的类测试的步骤

理论上，对象的状态空间是对象所有数据成员定义域的笛卡儿乘积。当对象含有多个数据成员时，对对象所有的可能状态进行测试是不现实的，这就需要对对象的状态空间进行简化，同时也要考虑对测试空间的全面覆盖。简化对象状态空间的基本思想类似于黑盒测试中常用的划分等价类的方法。依据软件设计规范或分析程序源代码，可以从对象数据成员的取值域中找到一些特殊值和一般性的区间。

例如：① char：name

特殊值：name＝＝null 一般区间：name！＝null

② int：allnum

特殊值：allnum＝＝0 一般区间：allnum＞0；allnum＜0

特殊值是设计规范里有特殊意义，在程序源代码里逻辑上需特殊处理的取值。位于一般性区间中的值不需要区别各个值的差别，在逻辑上以同样方式处理。

进行基于状态的测试时，首先要对受测试的类增加一些用于设置和检查对象状态的方法。通常是为每一个数据成员设置一个改变其取值的方法。另一项重要工作是编写作为主控的测试驱动程序，如果被测试的对象在执行某个方法时还要调用其他对象的方法，则需编写桩程序代替其他对象的方法。测试过程为：首先生成对象，接着向对象发送消息把对象状态设置到测试实例指定的状态，再发送消息调用对象的方法，最后检查对象的状态是否按预期的方式发生变化。

基于状态的测试步骤如下。

（1）根据设计文档或分析数据成员取值情况，导出对象逻辑空间，得到被测类的状态转移图。状态转移如图 4-8 所示。

A —a/b→ B

图 4-8　状态转移图

其中 A、B 表示两种状态，a 表示输入，b 表示输出。

（2）对受测类增加一些用于设置和检查对象状态的方法。

（3）对于每个状态，确定是哪些方法的合法起始状态（即允许何种操作）。

（4）从类中方法的调用关系图最下层开始，逐一测试类中方法。测试每个方法时，根据对象当前状态确定出对该方法的执行路径有特殊影响的参数。

4）基于状态的测试标准

进行对象状态测试时，根据希望测试出什么样的错误来选择相应的测试标准。对象

状态测试主要侧重于检测以下类型的错误。

(1) 状态错误：对象状态存在死状态、丢失状态和多余状态。死状态指在实际运行模型中对象进入此状态后就不会产生状态转移的状态；丢失状态指在构造模型中有而在实际运行模型中没有的状态；多余状态指在构造模型中没有而在实际运行模型中有的状态。

(2) 简单转移错误：对构造模型和实际运行模型相同的源状态进行同一简单转移，产生终态不同。

(3) 交叉转移错误：交叉转移引起的简单转移错误。

(4) 一般操作错误：两个模型中的相同操作引起的转移错误。

5) 基于状态的测试的优点及不足

基于状态的类测试方法的优势是可以充分借鉴成熟的有限状态机理论，但状态空间很大，执行起来还很难，不得不用自动化测试，而且测试覆盖率的计算不十分明确。使用基于状态的测试，主要检查行为和状态的改变，而不是内在逻辑，因此可能遗漏数据错误，尤其是没有定义对象状态的数据成员容易被忽略。而基于状态转移图的类测试技术难于描述：①继承的对象动态行为；②并发的动态行为；③由数据成员和成员函数构成的对象状态和对象状态转移。

4.3.3 UML 在类测试中的应用

1. 类在 UML 中的描述

UML 中的类表示为矩形框，主要由三部分组成，自上而下分别是类标题(如果是抽象类，这里的 Class 用斜体字表示)、类属性(Attributes)、操作(Operation，如果是抽象方法，Operation 用斜体字表示)，如图 4-9 所示。

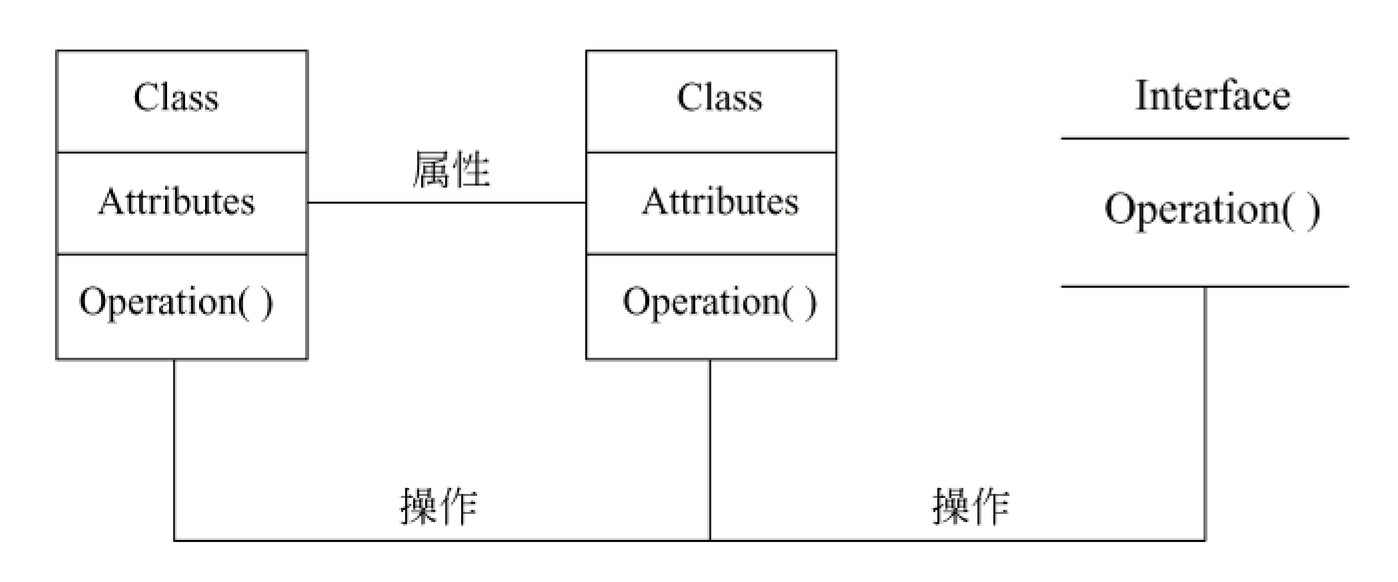


图 4-9 类的表述

2. 类的测试

通过一个例子来说明 UML 在类测试中的应用，如图 4-10 所示。

1) UML 中的类的描述

SignalLamp 是一个简单的信号灯类，包含了一个 String 类型的私有属性 state，用来存储信号灯的状态，3 个公共类方法分别是 getState()、setState()、changeState()，具体功能如下。

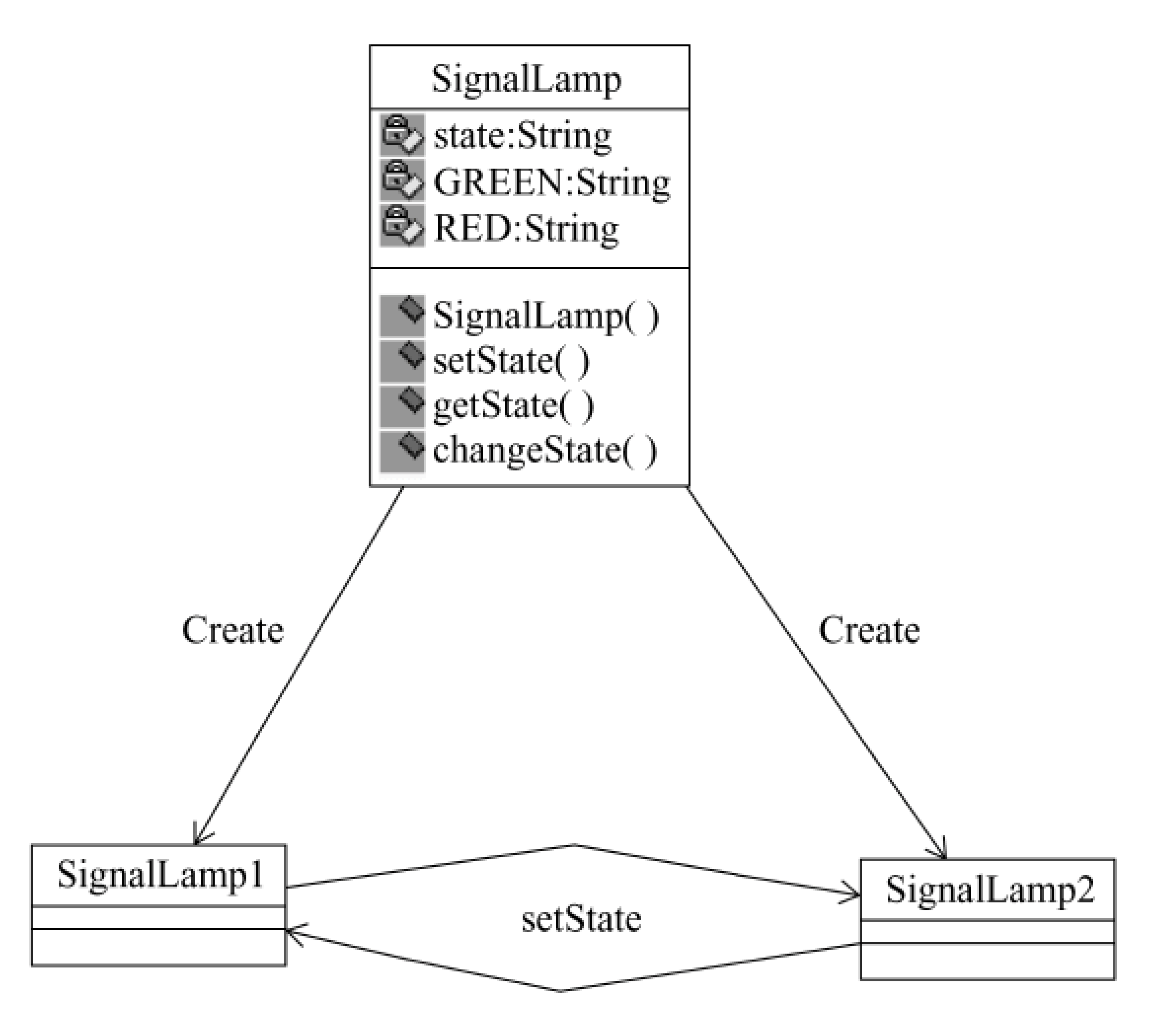


图 4-10　面向对象的实例

getState()：用来取信号灯的当前状态。

setState()：用来刷新信号灯的当前状态。

changeState()：用来验证输入的信号灯实例的状态。

2）测试驱动的构建

构建一个抽象类 TestCase,其行为类模型如图 4-11 所示。

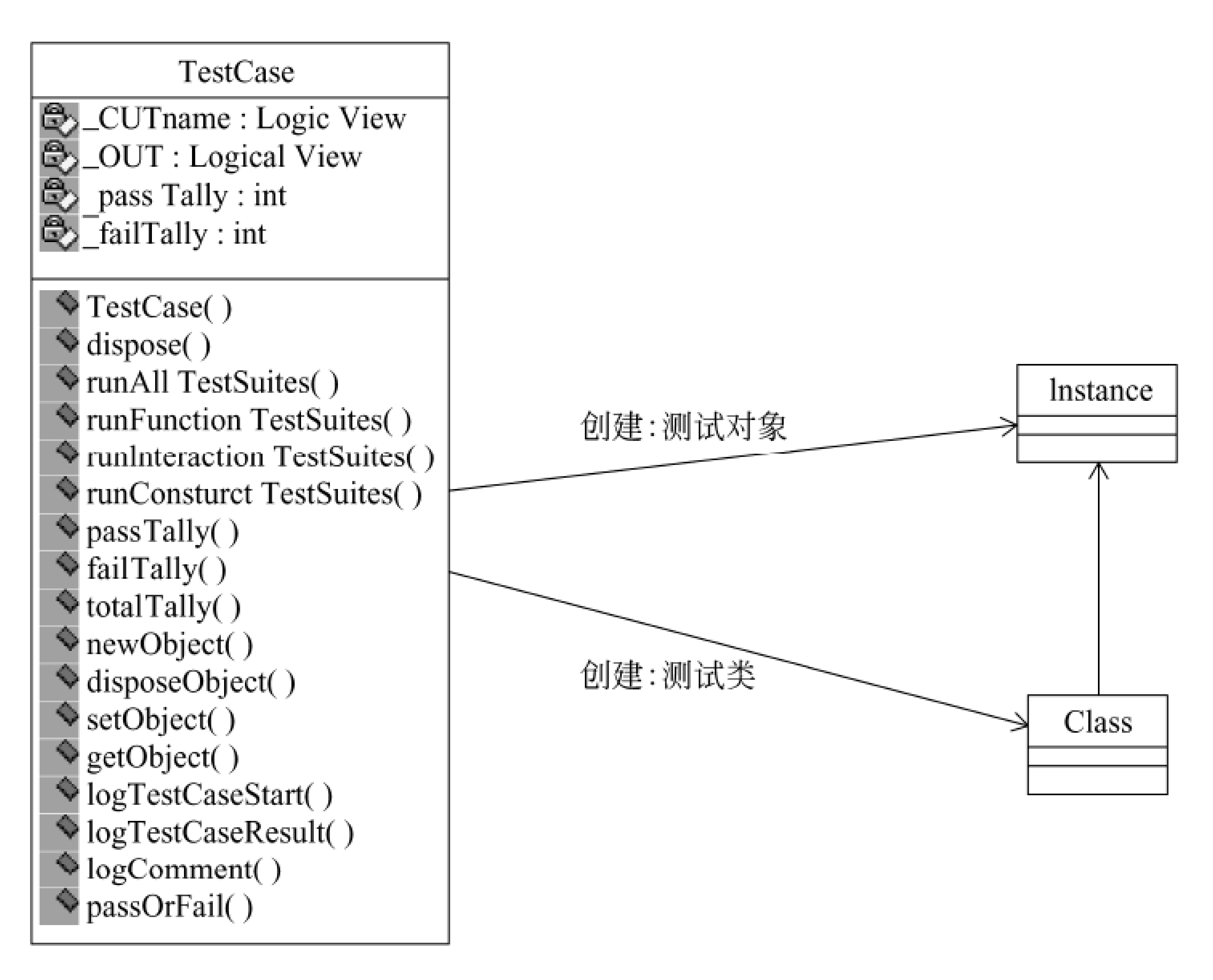


图 4-11　TestCase 类行为类模型

定义了 3 种测试接口分别来运行不同的测试方式,测试方式如下。

(1) 功能测试接口：如果测试用例根据类描述确定，那么这就是功能测试用例。

(2) 构造测试接口：如果测试用例着重测试类的构造(类可能包含多条构造方法)，那么这就是构造测试用例。

(3) 交互测试接口：如果测试用例测试发送消息对一个对象的操作是否正确，那么这就是交互测试用例。

确定这些接口是为了后续维护的方便，体现了遵循何种规则来确定测试用例并且关系到类的变化对测试用例的影响，表现了使用 TestCase 的行为实现。

3) 类测试的实现图

Lamp 类是一个接口类，定义了所有灯饰体的最高抽象描述，拥有两个接口方法 setState()和 getState()方法。不管什么类，只要实现了 Lamp 接口，就表明 Lamp 的具体可实现子类拥有了两个共有的可视行为"设置信号灯状态"和"取信号灯状态"。SignalLamp 是一个简单的信号灯类，并且该类在无参数构造时会产生一个 GREEN Signallamp 实例，它实现了 Lamp 接口，并且在 setState()和 getState()方法中具体实现方法代码，SignalLamp 类作为被测试对象，测试类关系图如图 4-12 所示。

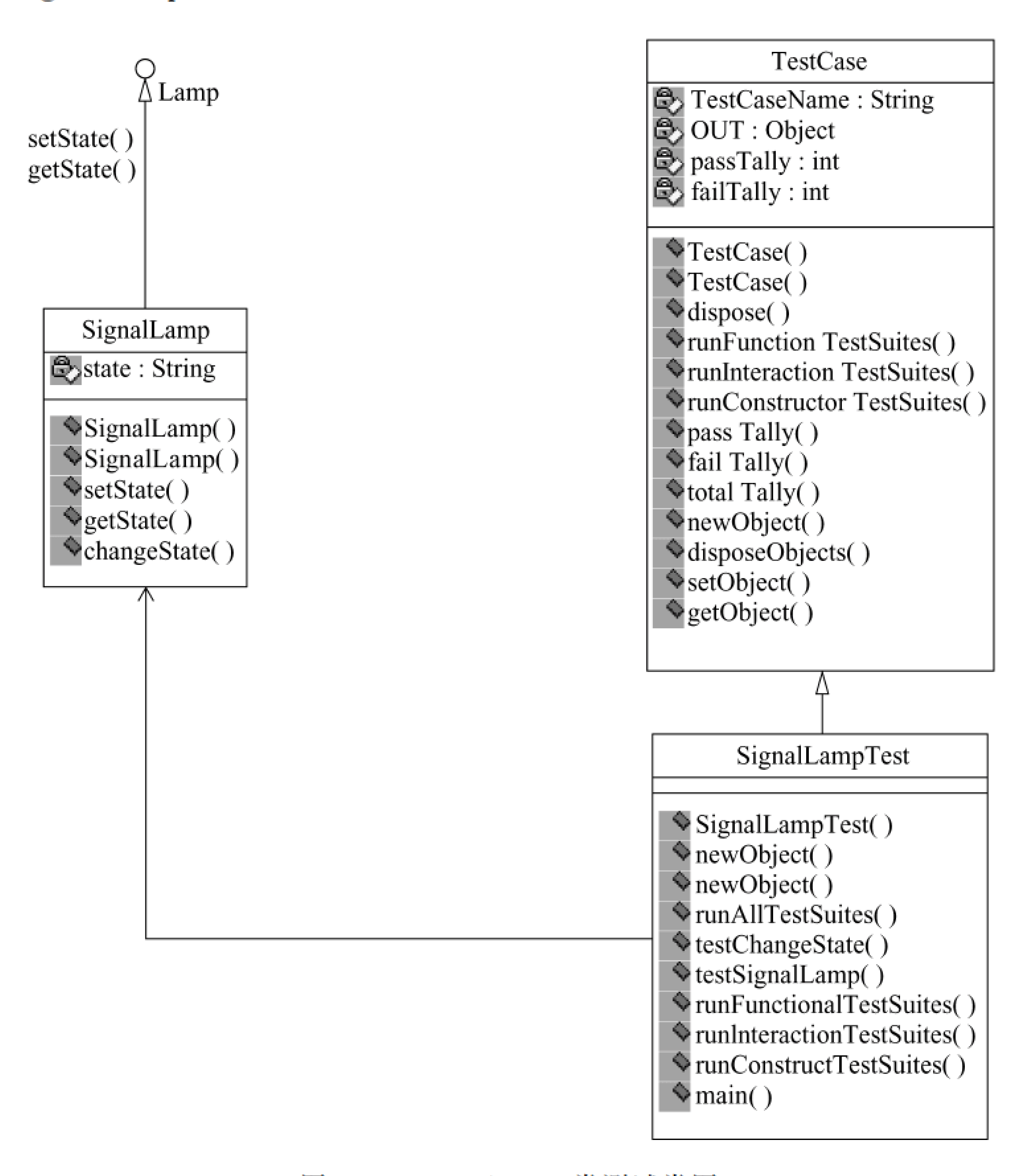


图 4-12 SignalLamp 类测试类图

4.4　面向对象的集成测试

类测试由验证类的实现是否和该类的说明完全一致的相关活动组成，如果类的实现正确，那么类的每个实例的行为也应该正确。

1. 与传统的集成测试的比较

传统的集成测试是由底向上通过集成完成的功能模块进行测试，一般可以在部分程序编译完成的情况下进行。而对于面向对象程序系统，相互调用的功能是散布在程序的不同类中的，类通过消息相互作用申请和提供服务。类的行为与它的状态密切相关，状态不仅仅体现在类数据成员的值，也许还包括其他类中的状态信息。由此可见，类之间的依赖关系很紧密，根本无法在编译不完全的程序上对类进行测试。所以，面向对象的集成测试通常需要在整个程序编译完成后进行。

在 OO 系统中，类之间的通信通过传递消息而完成，消息具有下述格式：

<实例名>.<方法名>.<变量>

有名称的实例调用具有指定名称的方法，或通过变量调用对象。在测试 OO 系统时，方法名没有唯一地确定控制流，这种函数和操作符随背景不同而变化，同一操作在不同的条件下行为各异的性质称为多态性。多态性推翻了代码覆盖和代码审查的传统定义。例如两个类 square 和 circle 都具有 area 方法，虽然具有相同的方法，但不同类的含义和调用方法背景不同，参数的含义也不同，对于 square 是边长，对于 circle 是半径。方法的行为也完全不同，也就是说，如果针对 square 测试了方法 area，并不意味着方法 area 对于 circle 也是正确的。

2. 面向对象的集成测试步骤

面向对象的集成测试能够检测出相对独立的类测试无法检测出的类相互作用时产生的错误。类测试保证成员函数行为的正确性，集成测试关注的是系统的结构和内部的相互作用。面向对象的集成测试可以分成两步进行：先进行静态测试，再进行动态测试。

静态测试主要针对程序的结构，检测程序结构是否符合设计要求。现在流行的一些测试软件都能提供一种可逆性工程的功能，即通过原程序得到类关系图和函数功能调用关系图，例如 International Software Automation 公司的 Panorama-2 for Windows 95、Rational 公司的 Rose C++ Analyzer 等，将可逆向工程得到的结果与 OOD 的结果相比较，检测程序结构和实现上是否有缺陷。换句话说，通过这种方法检测 OOP 是否达到了设计要求。

动态测试设计测试用例时，通常需要上述的功能调用结构图、类关系图或者实体关系图为参考，确定不需要被重复测试的部分，从而优化测试用例，减少测试工作量，使得进行的测试能够达到一定覆盖标准。测试所要达到的覆盖标准可以是：达到类所有的服务要求或服务提供的一定覆盖率；依据类间传递的消息，达到对所有执行线程的一定覆盖率；

达到类的所有状态的一定覆盖率;等等。同时也可以考虑使用现有的一些测试工具来得到程序代码执行的覆盖率。测试用例设计步骤如下。

(1) 首先选定测试类,参考 OOD 分析结果,确定出类的状态和行为、类或类成员函数间传递的消息、输入和输出的界定等。

(2) 确定覆盖标准。

(3) 利用结构关系图确定待测类的所有关联。

(4) 根据程序中类的对象构造测试用例,确认使用什么输入激发类的状态,使用类的服务和期望产生什么行为等。

在设计测试用例时,要设计确认类功能满足的输入,而且还要设计被禁止的例子,用于确认类的不合法的行为产生,例如与类状态不相适应的消息,以及不响应的服务等。

3. OO 软件的集成测试的策略

(1) 基于线程测试,基于线程的测试就是把对应一个输入或事件的类集合组装起来,也就是用响应系统的一个输入或一个事件的请求来组装类的集合。对每个线程都要分别进行组装和测试。

(2) 基于使用测试,基于使用的测试就是按分层来组装系统,可以先进行独立类的测试。在独立类测试之后,下一个类的层次叫从属类。从属类用独立类进行测试。这种从属类层的顺序测试直到整个系统被构造完成。传统软件使用驱动程序和连接程序作为置换操作,而 OO 软件不用。

(3) 对象交互是指一个对象(发送者)对另一个对象(接收者)的请求,发送者请求接收者执行接收者的一个操作,而接收者进行的所有处理工作就是完成这个请求。对象交互覆盖了 OO 程序中的绝大部分活动,包含了对象及其组件的消息,还包含了对象与其他相关对象之间的消息。对象是类的实例,通过类测试,说明类的实现是完整的,但对象交互将影响接收对象的内部状态。OO 系统集成时还必须进行类间合作关系的测试。

4. OO 集成测试的常用方法

穷举测试法可以达到 100%的覆盖率,是一种可靠的测试方法。但是,由于对象的交互作用的组合数量巨大,没有足够的时间构建和完成这些测试,为此,可以采用下述方法。

1) 抽样测试

抽样测试能够从一组可能的测试用例选择一个测试系列。测试过程的目的在于定义感兴趣的测试总体,然后定义一种方法,以便在这些测试用例中选择哪些被构建、哪些被执行。

2) 正交阵列测试

正交阵列测试提供了一种特殊的抽样方法,这种方法通过定义一组交互对象的配对方式组合,并尽力限制测试配置的组合数目倍增。正交阵列是一个数值矩阵,其中的每一列表示一个变量,例如,假设有 3 个变量 A、B、C,每个变量有 1、2、3 共 3 个级别,那么就有 27 种可能的组合。如果仅考虑配对组合,即一个给定级别仅出现两次,那么就只有表 4-3 所示的 9 种情况。

表 4-3　3 个变量、3 个级别的配对方式的组合情况

	1	2	3	4	5	6	7	8	9
A	1	1	1	2	2	2	3	3	3
B	1	2	3	1	2	3	1	2	3
C	3	2	1	2	1	3	1	3	2

5. 注意的问题

(1) OO 系统本质上是要通过较小的、可重用的组件构建的,因此,集成测试异常重要。

(2) OO 系统的底层组件的开发更具有并行性,因此,对频繁集成要求较高。

(3) 由于并行性高,集成测试时需要考虑类的完成顺序,也需要设计桩模块和驱动模块来模拟还没有完成的类的功能。

4.5　面向对象的系统测试

通过类测试和集成测试,仅能保证软件开发的功能得以实现,但不能确认系统运行时是否满足用户的需要,是否在实际使用中发生错误。为此,对完成开发的软件系统必须经过系统测试。通过系统测试可以检试系统中各部分配套运行的表现,以保证在工作的环境下系统各部分协调而正常工作。系统测试是独立于系统实现的,系统测试员不需知道实现采用的是过程代码还是面向对象代码。

系统测试时应该尽量搭建与用户实际使用环境相同的测试平台,对临时没有的系统设备部件,也应有相应的模拟手段。在系统测试时,应该参考 OOA 分析的结果,对应描述的对象、属性和各种服务,检测软件是否能够完全达到要求。系统测试不仅是检测软件的整体行为表现,也是对软件开发设计的再确认。系统测试的主要范围如下所述。

(1) 功能测试:测试是否满足开发要求,是否能够提供设计所描述的功能,是否用户的需求都能得到满足。功能测试是系统测试最常用和必需的测试,通常还会以正式的软件说明书为测试标准。

(2) 强度测试:测试系统的能力最高实际限度,即软件在一些超负荷的情况下功能是否可以实现的情况。如要求软件某一行为的大量重复、输入大量的数据或大数值数据、对数据库大量复杂的查询等。

(3) 性能测试:测试软件的运行性能。这种测试常常与强度测试结合进行,需要事先对被测软件提出性能指标,如传输连接的最长时限、传输的错误率、计算的精度、记录的精度、响应的时限和恢复时限等。

(4) 安全测试:验证安装在系统内的保护机构确实能够对系统进行保护,使之不受各种类型的干扰。安全测试时需要故意设计一些测试用例来试图突破系统的安全保密措施,检验系统是否有安全保密的漏洞。安全测试需要执行足够多的测试用例,以便对每个安全类别测试至少一个用户。进行安全测试需要考虑下述策略。

- 系统具有允许授权者访问和阻止非授权者访问的能力。
- 系统具有阻止非授权者访问其他不相关的系统资源的能力。

(5) 恢复测试：采用人工的干扰使软件出错，中断使用，检测系统的恢复能力，特别是通信系统。恢复测试时，应该参考性能测试的相关测试指标。

(6) 可用性测试：测试用户是否能够满意使用。具体体现为操作是否方便，用户界面是否友好等。

(7) 安装/卸载测试：安装测试就是要确保用在系统中的软件包能够提供足够的安装步骤，使得产品在工作条件下可以交付使用。对于可配置系统和与环境动态交互系统来说，安装测试异常重要。对于安装测试，要设计完全安装测试用例和定制安装测试用例。卸载测试主要测试系统能否正确地被卸载。

4.6 面向对象测试与传统测试的比较

传统的测试策略是从单元测试开始的，然后逐步进入集成测试，最后是确认和系统测试。单元测试集是最小的可编译程序单位，即子程序(如模块、子例程、过程)，当这些单元被独立测试之后，就被集成，而通过一系列的回归测试可以发现由于模块的接口和新单元的加入导致的副作用所带来的错误，最后，系统被作为一个整体进行测试以发现在需求中的错误。

面向对象软件测试运用面向对象技术，进行以对象概念为中心的软件测试。封装性、继承性、多态性和动态绑定性等面向对象特征的引入，增加了测试的复杂性。软件测试层次是基于测试复杂性分解的思想构造的，是软件测试的一种基本模式。面向对象软件测试呈现从单元级、集成级到系统级的分层测试，测试集成的过程是基于可靠部件组装系统的过程。测试可用不同的方法执行，通常的方法是按设计和实现的反向次序测试，首先验证不同层，然后使用事件集成不同的程序单元，最终验证系统级。根据测试层次结构确定相应的测试活动，并生成相应的层次。由于面向对象软件从宏观上来看是各个类之间的相互作用，因此，提出的测试方法中将对类层的测试作为单元测试，而对于由类集成的模块测试作为集成测试，系统测试与传统测试相同。

软件的质量不仅体现在程序的正确性上，也与所做的需求分析、软件设计密切相关。对已有的错误，往往不能通过简单的修补来纠正，因为可能会诱发更多错误，而必须追溯到软件开发的最初阶段。因此，为了保证软件的质量，应该考虑整个软件的生存期，特别是编码以前的各开发阶段的工作。于是，软件测试的实施范围必须扩充到包括在整个开发过程中各阶段的复查、评估和检测。

在整个软件生存期，确认、验证、测试分别有其侧重的阶段。确认主要体现在计划阶段、需求分析阶段，也会出现在测试阶段；验证主要体现在设计阶段和编码阶段；测试主要体现在编码阶段和测试阶段。事实上，确认、验证、测试是相辅相成的。确认无疑会产生验证和测试的标准，而验证和测试通常会在系统测试阶段帮助完成一些确认。

传统的测试计算机软件的策略是从单元测试开始的，然后逐步进入集成测试，最后是有效性和系统测试。单元测试集中在最小的可编译单位（如模块、子例程、进程)中，一旦

这些单元均被独立测试后，将被集成在程序结构中，这时要进行一系列的回归测试以发现由于模块的接口所带来的错误和新单元加入所导致的副作用，最后，系统被作为一个整体测试以保证发现在需求中的错误。

面向对象程序的结构不再是传统的功能模块结构，作为一个整体，原有集成测试所要求的逐步将开发的模块搭建在一起进行测试的方法已成为不可能。而且，面向对象软件抛弃了传统的开发模式，对每个开发阶段都有不同于以往的要求和结果，已经不可能用功能细化的观点来检测面向对象分析和设计的结果。因此，传统的测试模型对面向对象软件已经不再适用。

传统软件测试用例的设计是从各个模块的算法细节得出的，而 OO 软件测试用例则基于操作序列，以实现对类的说明。

小　　结

随着面向对象技术的发展和面向对象软件的不断出现，软件测试员也面临了新的问题。面向对象测试的整体目标和传统软件测试的目标是一致的，但是，面向对象的测试在策略和技术上有很大的改变。测试的视角扩大到包括分析和设计模型的评审。此外，测试的焦点也从过程构件移向了类。

当面向对象的程序已经完成时，对每个类进行单元测试。类测试使用了一系列不同的方法：基于服务的测试，基于状态的测试，基于数据流的测试。应该设计相应的测试序列以保证相关的操作被处理。类的状态被检查以确定是否存在错误。

集成测试可使用基于线程或基于使用的策略来完成。基于线程的测试集成一组相互协作来对某输入或时间做出相应的类。基于使用的测试按层次构造系统，从那些不使用服务器类的类开始。

系统测试是面向黑盒的并通过与传统软件类似的黑盒方法来完成，在系统测试中具体测试步骤包括：功能测试、强度测试、性能测试、安全测试、恢复测试、可用性测试、安装/卸载测试。

习　题　4

1. 面向对象的测试层次为________、________和________。
2. 面向对象的开发模型分为________、________和________ 3 个阶段。
3. 创建一个类，并用 UML 中的类图来描述。
4. 举例说明传统的过程测试的方法只适用于类中方法的测试，不适用于类的整体测试。
5. 举例说明 OO 集成测试中的基于使用测试的策略。

第 5 章　测试的设计与实现

学习要点：

- ❖ 测试计划。
- ❖ 测试设计。
- ❖ 测试执行。
- ❖ 测试总结。

软件测试是检验系统可靠性的重要手段，系统在投入使用之前必须进行严格的测试，而且测试必须按照一定的方法、步骤和措施实施，才能达到提高系统可靠性的目的。

随着软件开发规模的增大、复杂程度的增加，以寻找软件中的错误为目的的测试工作就显得更加困难。然而，为了找出程序中尽可能多的错误，开发出高质量的软件产品，加强对测试工作的设计、组织和管理尤为重要。

从软件的生存周期来看，测试往往指对程序的测试，这样做的优点是被测对象明确，测试的可操作性较强。但是，由于测试的依据是规格说明书、设计文档和使用说明书，如果测试后发现设计有错误，则此时的修改代价也相当昂贵。因此，理想的方法应该是对软件的开发过程，按软件工程各阶段形成的结果，分别进行严格的审查。

测试贯穿于软件的整个生存周期，是一个系统过程。系统化的测试过程能够在软件发布前及早地发现更多的问题，从而可以以最小的代价更正问题。测试的基本内容包括测试计划、测试设计、测试执行和测试总结等内容。

5.1　测 试 计 划

系统的高可靠性是指系统在遇到故障时，能够尽量不受到影响，或者把影响降到最低，并能够迅速地自动修正某些故障而恢复正常运行。由此可以看出，系统的高可靠性是在系统的分析、设计、编码和实施的过程中，通过测试过程而实现。测试必须按照一定的方法、步骤和措施实施，以达到提高系统可靠性的目的。

5.1.1　设计测试计划的目的

设计测试计划是一项重要的工作，主要目的如下所述。

1. 指导软件测试

(1) 在测试的过程中,经常遇到一些问题而导致测试过程延误,如果提前做好防范,把其列入软件测试计划内,那么软件测试则会更为顺利地进行。

(2) 在测试计划内列出风险评估,对于解决或避开风险将有很大的帮助。

2. 促进彼此沟通

测试的重点会根据所测试的产品不同而有所变化,如果把这些内容都列入测试计划中,那么就会让所有的测试人员在所进行的测试方向上达成一定的共识,从而避免产生偏差,促进了测试人员之间的沟通。

3. 协助质量管理

测试计划可以让整体的软件测试采取系统化的方式来进行,从而使测试的管理更易进行。

5.1.2 测试方案的制定

1. 测试方案设计的步骤

设计测试方案的步骤如下。

(1) 模型化被测系统并分析其功能。

(2) 根据外部观察设计测试用例。

(3) 根据代码分析、用猜测和启发式的方式研究添加测试用例。

(4) 给出每一个测试用例的预期结果,或者选择一种方法评估测试用例是否通过测试。

测试设计方案完成后,就可将这些测试用例应用到被测系统。在系统测试中,可以通过使用测试工具等来实现测试,也可编写用于特定应用的测试驱动,并且将测试代码添加到应用系统中来实现。一个典型的测试工具在测试时将启动被测软件,设置其环境,进入预测状态,然后应用测试用例进行测试,最后评估输出结果和状态。

2. 执行测试方案步骤

执行测试方案的步骤如下。

(1) 建立一个被测软件,最低限度地在操作上可以检验各部分之间接口的测试用例集。

(2) 执行测试用例集,评价每一个测试结果是否通过。

(3) 使用一个覆盖工具,运行测试用例集来评价所报告的覆盖。

(4) 如果需要的话,进一步开发附加的测试用例,检测没被覆盖的代码。

(5) 如果满足覆盖目标并且所有测试都已通过,则可以停止测试。

其中的一些步骤并不都是必要的,不过必须能够运行至少一个测试用例集并评价其结果。覆盖是用一个给定的测试用例集运行被测软件,获得测试策略所要求的百分比。测试设计与执行最好能与应用的分析、设计以及代码的编写并行进行。

3. 测试设计的类型

测试设计的类型可以分为基于功能的、基于实现的、混合的和基于故障的测试设计4种类型。

1）基于功能的测试设计

根据一个单元、子系统或系统指定的或预期的功能来设计测试，它与黑盒测试设计相同。

2）基于实现的测试设计

根据对源代码的分析来开发测试用例，它与白盒测试设计相同。

3）混合的测试设计

将基于功能的和基于实现的测试设计结合在一起，称为混合的测试设计，又称为灰盒测试。

4）基于故障的测试设计

有目的地在代码中设置故障，以便查看这些故障是否可以被测试软件所发现。此方法须根据系统的功能和特性、经费和时间等因素，选择不同的测试方案，进而选择不同的测试类型。

5.1.3 测试策略的制定

测试策略描述整体测试和每个阶段的测试方法。策略的制定是一项复杂的工作，需要由经验丰富的测试员来做，因为这将决定测试工作的成败。主要应包括整体测试策略、测试范围、测试风险分析和测试自动化策略等。

1. 软件整体测试策略

软件整体测试策略一般包含下列内容。

(1) 测试开始于单元级，然后延伸到整个系统中。

(2) 不同的测试技术适用于不同的时间点。

(3) 测试由软件的开发人员和独立测试组织来管理。

(4) 测试和调试是不同的活动，但是调试必须能够适应任何的测试策略。

测试策略描述测试过程的总体方法和目标，例如描述目前在进行哪一阶段的测试(单元测试、集成测试、确认测试、系统测试)以及每个阶段内在进行的测试种类(功能测试、性能测试、覆盖测试等)。测试策略必须提供能够用来检验某段源代码是否得以正确实现的低层测试，同时也要提供能够验证整个系统的功能是否符合用户需求的高层测试。一种策略必须为使用者提供指南，并且为管理者提供一系列的重要的里程碑。测试策略的制定是在软件的最终发布期已经确定后才开始进行的，所以测试的进度必须可测量，使得系统问题尽早暴露。

传统的测试策略是指定范围(如单元测试、集成测试、确认测试或系统测试)，并且指定按白盒或黑盒测试技术。尽管这些分类方法在某些情况下被证明是有效的，但它们并不适合于面向对象的系统。另外，单元测试和集成测试的分离将导致对面向对象的开发工作的不自然划分。

2. 单元测试的策略

单元测试的策略与集成测试不同。在为模块设计测试用例时，可以直接参考模块的源程序。所以单元测试的策略可以结合运用白盒测试法和黑盒测试法。具体做法有两种。

(1) 先仿照上述步骤用黑盒测试法提出一组基本的测试用例，然后用白盒测试法作验证。如果发现用黑盒测试法产生的测试用例未能满足所需的覆盖标准，就用白盒测试法补充新的测试用例来满足它们。覆盖的标准应该根据模块的具体情况确定。对可靠性要求较高的模块，通常要满足条件组合覆盖或路径覆盖标准。

(2) 先用白盒测试法分析模块的逻辑结构，提出一批测试用例，然后根据模块的具体功能用黑盒测试法进行补充。

3. 集成测试及其以后的测试阶段的策略

集成测试及其以后的测试阶段的策略一般采用黑盒方法，主要包括以下步骤。

(1) 用边界值分析法和(或)等价类划分法提出基本的测试用例。

(2) 用错误猜测法补充新的测试用例。

(3) 如果在程序的功能说明中含有输入条件的组合，则在测试一开始就使用因果图法。然后再按以上(1)、(2)步骤进行。

5.1.4　测试计划的制定

完善的测试计划是进行测试的基础，而测试的质量直接导致开发系统的产品质量。完成一个测试需要多个步骤，如选择测试策略、执行测试需求、问题跟踪报告等。这些步骤都是相互独立的，但它们又是相互关联和相互影响的。因此，测试者必定要有一个能够起到总体框架作用的测试计划，才能使测试有条不紊地进行。测试计划应该作为测试的起始步骤和重要环节，是对测试工作的总体描述。

1. 测试计划的定义

测试计划明确了预定的测试活动的范围、途径、资源及进度安排的文档，并确认了测试项、被测特征、测试任务、人员安排以及任何突发的风险。

2. 测试计划的内容

测试计划的主要内容如下所述。

1) 测试项目简介

(1) 归纳所要求测试的软件项和软件特性，可以包括系统目标、背景、范围及引用材料等。

(2) 在高层测试计划中，如果存在下述文件，则需要引用它们：项目计划、质量保证计划、有关的政策、有关的标准等。

2) 测试项

描述被测试的对象，包括其版本、修订级别，并指出在测试开始之前对逻辑关系或物理变换的要求。

3）被测试的特性

指明所有要测试的软件特性及其组合，指明每个特性或特性组合有关的测试设计说明。

4）不被测试的特性

指出不被测试的所有特性和特性的有意义的组合及其理由。

5）测试方法

（1）描述测试的总体方法，规定测试指定特性组合需要的主要活动和时间。

（2）规定所希望的测试程度，指明用于判断测试彻底性的技术，例如检查哪些语句至少执行过一次。

（3）指出对测试的主要限制，例如测试项可用性、测试资源的可用性和测试截止期限等。

6）测试开始条件和结束条件

（1）规定各测试项在开始测试时需要满足的条件。

（2）测试通过和测试结束的条件。

7）测试提交的结果与格式

指出测试结果及显示的格式。

8）测试环境

（1）测试的操作系统和需要安装的辅助测试工具（来源与参数设置）。

（2）软件、硬件和网络环境设置。

9）测试者的任务、联系方式与培训

（1）测试成员的名称、任务、电话、电子邮件等联系方式。

（2）为完成测试需要进行的项目课程培训。

10）测试进度与跟踪方式

（1）在软件项目进度中规定的测试里程碑以及所有测试项传递时间。

（2）定义所需的新的测试里程碑，估计完成每项测试任务所需的时间，为每项测试任务和测试里程碑规定进度，对每项测试资源规定使用期限。

（3）报告和跟踪测试进度的方式：每日报告、每周报告、书面报告、电话会议等方式。

11）测试风险与解决方式

（1）预测测试计划中的风险。

（2）规定对各种风险的应急措施（延期传递的测试项可能需要加班、添加测试人员或是减少测试内容）。

12）测试计划的审批和变更方式

（1）审批人和审批生效方式。

（2）如何处理测试计划的变更。

3. 测试计划的层次

一般而言，测试计划可分为3个层次。

1）概要测试计划

概要测试计划是软件项目实施计划中的一项重要内容，应当在软件开发初期，即需求

分析阶段制定。这项计划应当定义测试对象和测试目标,确定测试阶段和测试周期的划分,制定测试人员、软硬件资源和测试进度等方面的计划,规定软件测试方法、测试标准以及支持环境和测试工具。例如,被测试程序的语句覆盖率要达到 95%;第三级以上的错误修复率需要达到 95%;所有决定不修复的轻微错误都必须经过专门的质量评审委员会同意;等等。

2）详细测试计划

详细测试计划是针对子系统在特定的测试阶段所要进行的测试工作制定出来的详细计划。它详细规定了测试小组的各项测试任务、测试策略、任务分配和进度安排等。

3）测试实施计划

测试实施计划是根据详细测试计划制定的测试者的测试具体实施计划。它规定了测试者在每一轮测试中负责测试的内容、测试强度和工作进度等。测试实施计划是整个软件测试计划的组成部分,是检查测试实际执行情况的重要依据。

4. 测试计划举例

图 5-1 所示的是一份测试计划目录。

1. 总论
    - 1.1 项目背景
    - 1.2 项目目标
    - 1.3 系统视图
    - 1.4 文档目标
    - 1.5 文档摘要
2. 测试策略
    - 2.1 总体策略
    - 2.2 测试范围
    - 2.3 风险分析
3. 测试方法
    - 3.1 里程碑技术
    - 3.2 测试用例设计
    - 3.3 测试实施过程
    - 3.4 测试通过标准
    - 3.5 测试挂起标准
4. 测试组织
    - 4.1 测试团队结构
    - 4.2 功能划分
    - 4.3 联系方式
5. 资源需求
    - 5.1 培训需求
    - 5.2 硬件需求
    - 5.3 软件需求
    - 5.4 办公室空间需求
    - 5.5 相关信息保存位置
6. 时间进度安排
7. 测试过程管理
    - 7.1 测试文档
    - 7.2 缺陷处理过程

图 5-1 测试计划目录

5.1.5 测试的组织

为了尽可能多地找出程序中的错误,生产出高质量的软件产品,加强对测试工作的组织和管理就显得尤为重要。

- 测试的组织方式是小组。
- 测试内部的个体分为测试人员和支持人员(管理人员属于支持人员)。
- 测试的工作实体(最小组织单位)是测试小组和支持小组,分别由小组长全权负责。小组长向测试主管负责。

测试小组根据测试项目或评测项目的需要临时组建,小组长也是临时指定的。与项目组的最大区别是生存周期短,一般是 2 周～4 个月。在系统测试期间或系统评测期间,测试组长是测试对外(主要是指项目组之外的事务)的唯一接口,对内完全负责组员的工作安排、工作检查和进度管理。

支持小组按照内部相关条例负责测试的后勤保障和日常管理工作,机构设置一般相对比较稳定。主要负责网络管理、数据备份、文档管理、设备管理和维护、员工内部培训、测试理论和技术应用、日常事务管理和

检查等。

另外，测试对于每一个重要的产品方向，均设置1～3个人长期研究和跟踪竞争对手的产品特征、性能、优缺点等。在有产品测试时，指导或参加测试(但不一定作为测试组长)，尤其是在需求分析阶段多多参与；在没有产品测试时，进行产品研究，并负责维护和完善测试设计。

1. 测试的组织步骤

测试的组织步骤如下。

(1) 测试人员要仔细阅读有关资料，包括规格说明、设计文档、使用说明书及在设计过程中形成的测试大纲、测试内容及测试的通过准则，全面熟悉系统，编写测试计划，设计测试用例，做好测试的准备工作。

(2) 为了保证测试的质量，将测试过程分成5个阶段：代码审查、单元测试、集成测试、确认测试和系统测试。

(3) 代码会审。代码会审是由一组人通过阅读、讨论和争议对程序进行静态分析的过程。会审小组在充分阅读待审程序文本、控制流程图及有关要求、规范等文件基础上，召开代码会审会，程序员详细讲解程序的逻辑，并展开热烈的讨论，以提示错误的关键所在。实践表明，程序员在讲解过程中能发现许多原来没有发现的错误，而讨论则促使了问题的暴露。

(4) 单元测试。单元测试集中在检查软件设计的最小单位上，测试发现实现该模块的实际功能与定义该模块的功能说明不符合的情况，以及编码的错误。

(5) 集成测试。集成测试是将模块按照设计要求组装起来同时进行测试，主要目标是发现与接口有关的问题。如数据穿过接口时可能丢失；一个模块与另一个模块可能由于疏忽而造成有害影响；把子功能组合起来可能无法产生预期的主功能；可以接受的个别误差可能积累到不能接受的程度；全程数据结构可能有误；等等。

(6) 确认测试。确认测试的目的是向未来的用户表明系统能够像预定要求那样工作。经集成测试后，已经按照设计把所有的模块组装成一个完整的软件系统，接口错误也已经基本排除。接着应该进一步验证软件的有效性，这就是确认测试的任务，即确定软件的功能和性能达到用户要求。

(7) 系统测试。软件开发完成以后，最终还要与系统中其他部分配套运行，进行系统测试。包括恢复测试、安全测试、强度测试和性能测试等。

2. 测试的人员组织

为了保证软件的开发质量，软件测试应贯穿于软件定义与开发的整个过程。因此，对分析、设计和实现等各阶段所得到的结果，包括需求规格说明、设计规格说明及源程序都应进行软件测试。基于此，测试人员的组织也应是分阶段的。

(1) 软件的设计和实现都是基于需求分析规格说明进行的。需求分析规格说明是否完整、正确、清晰是软件开发成败的关键。为了保证需求定义的质量，应对其进行严格的审查。

(2) 设计评审。软件设计是将软件需求转换成软件表示的过程。主要描绘出系统结

构、详细的处理过程和数据库模式。按照需求的规格说明对系统结构的合理性、处理过程的正确性进行评价，同时利用关系数据库的规范化理论对数据库模式进行审查。

(3) 程序的测试。程序的测试是整个软件开发过程中交付用户使用前的最后阶段，是软件质量保证的关键。软件测试在软件自下而上周期中横跨两个阶段。通常在编写出每一个模块之后，就对它进行必要的单元测试。编码与单元测试属于软件自下而上周期中的同一阶段，该阶段的测试工作由编程组内部人员进行交叉测试(避免编程人员测试自己的程序)。这一阶段结束后，进入软件自下而上周期的测试阶段，对软件系统进行各种综合的测试。测试工作由专门的测试组完成，负责整个测试的计划、组织工作。测试组的其他成员由具有一定的分析、设计和编程经验的专业人员组成，人数根据具体情况可多可少，一般 3～5 人为宜。

3. 软件测试文件

软件测试文件描述要执行的软件测试及测试的结果。由于软件测试是一个很复杂的过程，同时也是软件开发工作中的一个阶段，对于保证软件的质量和运行有着重要意义，必须把对它们的要求、过程及测试结果以正式的文件形式写出。测试文件的编写是测试工作规范化的一个组成部分。

测试文件不只在测试阶段才考虑，因为测试文件与用户有着密切的关系，它在软件开发的需求分析阶段就开始着手。在设计阶段的一些设计方案也应在测试文件中得到反映，以利于设计的检验。测试文件对于测试阶段工作的指导与评价作用更是非常明显的。需要特别指出的是，在已开发的软件投入运行的维护阶段，常常还要进行再测试或回归测试，这时仍须用到测试文件。

1) 测试文件的类型

根据测试文件所起的作用不同，通常把测试文件分成两类，即测试计划和测试分析报告。测试计划详细规定测试的要求，包括测试的目的、内容、方法、步骤以及测试的准则等。由于要测试的内容可能涉及软件的需求和设计，因此必须及早开始测试计划的编写工作。不应该在着手测试时，才开始考虑测试计划。通常，测试计划的编写从需求分析阶段开始，到软件设计阶段结束时完成。测试报告是对测试结果的分析说明，经过测试后，证实了软件符合需求的能力，以及它的缺陷和限制，并给出评价的结论性意见，这些意见既是对软件质量的评价，又是决定该软件能否交付用户使用的依据。由于要反映测试工作的情况，自然要在测试阶段内编写。

2) 测试文件的使用

测试文件的重要性表现在以下几个方面。

(1) 验证需求的正确性：测试文件中规定了用以验证软件需求的测试条件，研究这些测试条件对弄清用户需求的意图是十分有益的。

(2) 检验测试资源：测试计划不仅要用文件的形式把测试过程规定下来，还应说明测试工作中需要的资源，进而检验这些资源是否可以得到，以及它的可用性如何。如果某个测试计划已经编写出来，但所需资源仍未落实，那必须及早解决。

(3) 明确任务的风险：有了测试计划，就可以弄清楚测试可以做什么，不能做什么。了解测试任务的风险有助于对潜伏的可能出现的问题事先做好思想上和物质上的准备。

(4) 生成测试用例：测试用例的好坏决定着测试工作的效率，选择合适的测试用例是做好测试工作的关键。在测试文件编制过程中，按规定的要求精心设计测试用例有重要的意义。

(5) 评价测试结果：测试文件包括测试用例，即若干测试数据及对应的预期测试结果。完成测试后，将测试结果与预期的结果进行比较，便可对已进行的测试提出评价意见。

(6) 再测试：测试文件中规定和说明的内容对在之后的维护阶段由于各种原因而增加的需求进行再测试时是非常有用的。

(7) 决定测试的有效性：完成测试后，把测试结果写入文件，这对分析测试的有效性，甚至整个软件的可用性提供了依据，同时还可以证实有关方面的结论。

(8) 测试文件的编制：在软件的需求分析阶段，就开始测试文件的编制工作，各种测试文件的编写应按一定的格式进行。

5.2 测试设计

测试设计是一种特殊的软件系统的设计和实现，它是通过执行另一个以发现错误为目标的软件系统来实现的。测试设计过程输出的是各测试阶段使用的测试用例。

将在测试计划阶段制定的测试活动分解，进而细化为若干个可执行的子测试过程，构造出测试计划中说明的执行测试所需的要素，这些要素通常包括驱动程序、测试数据集和实际执行测试所需的软件；同时为每个测试过程选择适当的测试用例，准备测试环境和测试工具。

测试设计是使用一个测试策略产生一个测试用例集的过程。

测试设计涉及以下 3 个问题。

(1) 有意义的测试点的识别。

(2) 将这些测试点放入一个测试序列。

(3) 为序列中的每个测试点定义预期的结果。

测试点是软件系统中一个可独立测试的功能或模块，测试用例是被测软件的具体测试步骤和预期结果的集合。

5.2.1 建立测试配置

1. 测试配置的内容

测试配置是实现测试的必要条件，在项目进行期间，测试所用到的任何配置资源都要考虑到。测试配置的内容一般包括以下一些。

(1) 人员：人数、经验和专长，全职、兼职或学生。

(2) 设备：计算机、打印机等。

(3) 测试环境：硬件、软件环境。

(4) 测试工具：黑盒或白盒测试工具。

(5) 办公室或实验室：地点、大小等。

(6) 专业测试公司：是否需要，费用如何。

(7) 其他需求：移动存储器、电话、通信等。

具体的要求取决于项目、小组和公司，配置测试资源需要仔细完成，开始计划不好，到项目后期获取资源通常很困难，甚至无法做到，因此创建完整的测试配置是不容忽视的。

2. 测试环境配置

测试环境配置与测试直接相关。配置测试环境是测试实施的一个重要阶段，测试环境适合与否会严重影响测试结果的真实性和正确性。测试环境包括硬件环境和软件环境。硬件环境指测试必需的服务器、客户端、网络连接设备，以及打印机/扫描仪等辅助硬件设备所构成的环境；软件环境指被测软件运行时的操作系统、数据库及其他应用软件构成的环境。

1) 环境配置过程中遇到的问题

建立一个测试平台很容易，但是要建立一个符合产品需求的测试环境相当困难。在建立测试环境时通常遇到下述困难。

(1) 资源不足。建立一个完善的测试环境需要足够的预算来支持，对于一些小型的企业来说，由于实际状况的制约，经常使用一些已经被淘汰的机器来组成测试环境。就软件测试的观点而言，在测试环境不齐全的情况下所进行的软件测试，大部分只能进行功能层面的测试。

(2) 操作系统更新。操作系统的种类繁多，特别是硬件性能相当优异，以目前的硬件设备不仅可以安装个人操作系统还可以安装服务器级的操作系统。操作系统不仅有版本的更新，还有修正版本更新。这样一来，对配置环境来说无疑是异常注意。

(3) 硬件设备的更新。硬件设备的更新速度与软件相比非常快，对于配置环境来说这不仅影响投资，还影响成果。例如，目前所投资测试环境的硬件是最新的设备，可是到了明年这些设备就成了过时的产品，到时候摆在面前的问题就是：要淘汰目前的设备更换新的设备，还是部分硬件升级其他保持不变。无论决定使用哪种方式，这样的预算花费及所需的人力都属于投资的一部分。

(4) 新的软件的不断推出。软件产业与硬件产业一样，有版本更新的影响因素存在。如现在所开发的软件与某一软件有依存关系存在，这样的依存关系会导致只要有新版本的软件出现或修正版推出，测试人员就必须安装新的版本做测试。也就是说，只要有新的版本，测试的广度就会增加。

(5) 客户端复杂的使用环境。只要产品交到客户手上，使用权就在客户，所以为了确保使用者能够正确地使用，最好的办法就是在产品开发过程中做好使用规划。

综上所述，创造出好的测试环境难度很大，而且投资也不少。但是测试环境的配置是必需的，也就是说，在一个软件上市之前，进行软件测试必不可少。它能保证软件的质量，降低产品的退货率，从而提升产品的总体收入。

2) 软件环境

在实际测试中，软件环境又可分为主测试环境和辅测试环境。

(1) 主测试环境。主测试环境是测试软件功能、安全可靠性、性能、易用性等指标的

主要环境。一般来说，配置主测试环境可遵循下列原则。

- 符合软件运行的最低要求，测试环境首先要保证能支撑软件正常运行。
- 选用比较普及的操作系统和软件平台。
- 构造相对简单、独立的测试环境。除了操作系统，测试机上只安装软件运行和测试必需的软件，以免不相关的软件影响测试实施。
- 无毒的环境。利用有效的正版杀毒软件检测软件环境，保证测试环境中没有病毒。

(2) 辅测试环境。辅测试环境常常用来满足不同的测试需求或特殊测试项目。

- 兼容性测试：在满足软件运行要求的范围内，可选择一些典型的操作系统和常用主要软件对其安装、卸载和主要功能进行验证。
- 模拟真实环境测试：有些软件，特别是面向大众的商品化软件，在测试时常常需要考察其在真实环境中的表现，如测试杀毒软件的扫描速度时，硬盘上布置的不同类型文件的比例要尽量接近真实环境，这样测试出来的数据才有实际意义。
- 横向对比测试：利用辅测试环境模拟出完全一致的测试环境，从而保证各个被测软件平等对比。

测试环境的配置对于保证测试的正确性非常重要，因此，在测试中的测试环境是如何配置的，必须给予说明。

5.2.2 测试用例设计

软件测试也是一种工程，也就是说，需要以工程的角度去认识软件测试、以工程的方法去完成软件测试工作。在测试之前，需要明确测试的内容以及如何完成对这些内容的测试，即通过设计测试用例来实现软件测试。

1. 为什么需要测试用例

在软件测试中，测试用例的作用如下。

(1) 在进行软件测试时，可以将部分测试工作外包，并要求外包人员根据所设计的测试用例进行测试。这样做可以节省测试人员的数量。

(2) 当管理者不知道软件测试需要多长时间时，可以通过测试用例的种类和数量来估算所需要的时间。

(3) 当测试人员不知道要求测试到何种程度时，可以根据测试用例，基于不同的状况来调整测试内容。

(4) 由于利用模块化的方式来归纳测试用例，所以测试人员可以知道所进行的测试是属于程序的哪个部分的。

(5) 可以根据测试用例的执行结果产生测试报告。

2. 测试用例的概念

测试用例是指为实施一次测试而向被测系统提供的输入数据、操作或各种环境设置，它是对测试流程中每个测试内容的进一步实例化，控制着软件测试的执行过程。测试用例就是以发现错误为目的而设计的一组测试数据和测试执行步骤。软件测试用例的主要

内容如下。

（1）测试索引：索引标识了测试需求，测试索引就是测试需求分析。

（2）测试环境：测试实施步骤所需的资源及其状态。

（3）测试输入：运行本测试所需的代码和数据，包括测试模拟程序和测试模拟数据。

（4）测试操作：在测试中所执行的具体操作。

（5）预期结果：是比较测试结果的基准。

（6）评价标准：指根据测试结果与预期结果的偏差，判断被测对象质量状态的依据。

测试索引和测试环境在测试需求分析步骤中定义，是软件测试计划的内容。测试输入、测试操作、预期结果和评价标准的描述性定义在软件设计步骤中定义，是软件测试说明的内容；测试输入、测试操作、预期结果和评价标准的计算机表示（代码/数据定义）在软件测试实现步骤中给出，是软件测试程序产品。

软件测试用例是软件测试结果的生成器，即每执行一次测试用例都产生一组测试结果。

一个典型的测试用例应该包括下列详细信息：测试目标、待测试的功能、测试环境及条件、测试日期、测试输入、测试步骤、预期的输出、评价输出结果的准则。所有的测试用例应该经过专家评审才可以使用。

3. 测试用例的类型

按测试目的不同，测试用例主要可分为以下几种类型。

1）等价类划分测试用例

（1）如果某个输入条件规定了取值范围或值的个数，则可确定一个合理的等价类（输入值或个数在此范围内）和两个不合理的等价类（输入值或个数小于这个范围的最小值或大于这个范围的最大值）。

（2）如果规定了输入数据的一组值，而且程序对不同的输入值做不同的处理，则每个允许输入值是一个合理等价类，此处还有一个不合理等价类（任何一个不允许的输入值）。

（3）如果规定了输入数据必须遵循的规则，可确定一个合理等价类（符合规则）和若干个不合理等价类（从各种不同角度违反规则）。

（4）如果已划分的等价类中各元素在程序中的处理方式不同，则应将此等价类进一步划分为更小的等价类。

（5）确定测试用例。

（6）为每一个等价类编号。

（7）设计一个测试用例，使其尽可能多地覆盖尚未被覆盖过的合理等价类。重复这步，直到所有合理等价类被测试用例覆盖。设计一个测试用例，使其只覆盖一个不合理等价类。

2）边界测试用例

边界测试是通过输入大小不同的数据和不正确的数据，观察被测程序是否产生错误的行为。边界测试用例是为了实现边界测试所构造的测试用例。使用边界值分析方法设计测试用例时一般与等价类划分结合起来。但它不是从一个等价类中任选一个例子作为代表的，而是将测试边界情况作为重点目标，选取正好等于、刚刚大于或刚刚小于边界值的测试数据。

(1) 如果输入条件规定了值的范围,可以选择正好等于边界值的数据作为合理的测试用例,同时还要选择刚好越过边界值的数据作为不合理的测试用例。如输入值的范围是[1,100],可取0、1、100、101等值作为测试数据。

(2) 如果输入条件指出了输入数据的个数,则按最大个数、最小个数、比最小个数少1、比最大个数多1等情况分别设计测试用例。例如,一个输入文件可包括1~255个记录,则分别设计有1个记录、255个记录以及0个记录的输入文件的测试用例。

(3) 对每个输出条件分别按照以上原则(1)或(2)确定输出值的边界情况。例如,一个学籍管理系统规定,只能查询2005~2008级大学生的各科成绩,可以设计测试用例,使得查询范围内的某一届或四届学生的学生成绩,还需设计查询2004级、2009级学生成绩的测试用例(不合理输出等价类)。

由于输出值的边界不与输入值的边界相对应,所以要检查输出值的边界不一定可能,要产生超出输出值之外的结果也不一定能做到,但必要时还需试一试。

(4) 如果程序的规格说明给出的输入或输出域是个有序集合(如顺序文件、线性表、链表等),则应选取集合的第一个元素和最后一个元素作为测试用例。

3) 功能测试用例

功能测试是软件测试最重要的一项测试,功能测试用例约占测试用例集的50%~80%,设计功能测试用例主要考虑以下4点。

(1) 功能是否符合要求。

(2) 功能是否完整。

(3) 功能是否有作用。

(4) 功能是否无错误。

4) 设置测试用例

测试代码的逻辑结构和使用的数据是否符合系统需求。

5) 压力测试用例

压力测试要找出安全临界值,压力测试用例必须设计出不同等级的压力环境来查看所测试的软件使用状况。压力环境的设置可以根据下述几点考虑。

(1) CPU处理速度。

(2) CPU使用率。

(3) 磁盘空间。

(4) 内存容量。

(5) 虚拟内存容量。

(6) 使用者数量。

(7) 处理信息量的多少。

6) 错误处理测试用例

设计错误处理测试用例就是尽量设计一些可以让被测软件发生或可能发生错误的环境来查看软件是否依然正常运行,设计错误处理测试用例考虑如下几点。

(1) 防范错误发生。

(2) 如何处理错误。

(3) 如何告知错误。

7）回归测试用例

回归测试的主要目的是确保被改动的程序达到了修改的目的，但不会引起其他问题的发生。最保险的回归测试策略是将所有的测试用例重新测试一遍，显然这也是最无效率的回归测试方式。回归测试与其他测试的最大区别是设计测试用例的考虑方向和测试用例的使用情况。测试用例的考虑方向是指必须确保核心功能和其他主要功能运作正常。测试用例使用情况是指测试用例可被重复使用和用户可累加。

8）状态测试用例

可用程序状态来表示所有的控制流程。测试的出发点是站在使用者角度，但每个使用者的习惯不同，所以设计状态测试用例必须从不同的层面进入。

9）结构测试用例

白盒测试是结构测试，所以被测对象基本上是源程序，以程序的内部逻辑为基础设计测试用例。当程序中有循环时，覆盖每条路径是不可能的，要设计使覆盖程度较高的或覆盖最有代表性的路径的测试用例。几种常用的覆盖包括语句覆盖、判定覆盖、条件覆盖、条件组合覆盖、路径覆盖等。

10）其他测试用例

还有一些测试用例也是经常使用的，例如性能测试用例、兼容测试用例、发行验证测试用例和使用界面测试用例等。

4. 设计测试用例的原则

设计测试用例的原则如下。

（1）一个好的测试用例能够发现之前没有发现的错误。

（2）测试用例应由测试输入数据和与之对应的预期输出结果这两部分组成。

（3）在设计测试用例时，应当包含合理的输入条件和不合理的输入条件。

5. 测试用例的策略与选择

（1）设计测试用例的规则和策略如下。

- 测试用例的代表性：能够代表各种合理和不合理的、合法和非法的、边界和越界的，以及极限的输入数据、操作和环境设置等。
- 测试结果的可判定性：测试执行结果的正确性是可判定的或可评估的。
- 测试结果的可再现性：对同样的测试用例，系统的执行结果应当是相同的。

测试用例的选择既要有一般情况，也应有极限情况以及最大和最小的边界值情况。使用测试的目的是暴露软件中隐藏的缺陷，所以在设计选取测试用例和数据时要考虑那些易于发现缺陷的测试用例和数据，结合复杂的运行环境，在所有可能的输入条件和输出条件中确定测试数据，来检查软件是否都能产生正确的输出。

（2）测试用例的设计方法不是单独存在的，具体到每个测试项目都会用到多种方法，每种类型的软件有各自的特点，每种测试用例设计的方法也有各自的特点，针对不同软件如何设计出全面的测试用例的问题非常重要。因此在实际测试中，联合使用各种测试方法，形成综合策略，通常先用黑盒测试法设计基本的测试用例，再用白盒测试法补充一些必要的测试用例。

(3) 在实际测试中,综合使用各种方法才能有效提高测试效率和测试覆盖度,这就需要掌握这些方法的原理,积累更多的测试经验,以有效提高测试水平。

(4) 首先进行等价类划分,包括输入条件和输出条件的等价划分,将无限测试变成有限测试,这是减少工作量和提高测试效率的有效方法。

(5) 在任何情况下都必须使用边界值分析方法。经验表明用这种方法设计出测试用例发现程序错误的能力最强。

(6) 对照程序逻辑,检查已设计出的测试用例的逻辑覆盖程度。如果没有达到要求的覆盖标准,应当再补充足够的测试用例。

(7) 对于业务流清晰的系统,可以利用场景法贯穿整个测试案例过程,在案例中综合使用各种测试方法。

6. 一个好的测试用例的特征

(1) 可以最大程度地找出软件隐藏的缺陷。

(2) 可以最高效率地找出软件缺陷。

(3) 可以最大程度地满足测试覆盖要求。

(4) 既不太复杂,也不能过分简单。

(5) 可以清楚地判定软件缺陷,包含以下两个原则。

- 测试用例包含期望的正确的结果。
- 待查的输出结果或文件必须尽量简单明了。

(6) 不包含重复的测试用例。

(7) 测试用例内容清晰、格式一致和分类组织。

7. 测试用例的设计

1) 测试用例文档

编写测试用例文档应有文档模板,须符合内部的规范要求。测试用例文档将受制于测试用例管理软件的约束。软件产品或软件开发项目的测试用例一般以该产品的软件模块或子系统为单位,形成一个测试用例文档,但并不绝对。

测试用例文档由简介和测试用例两部分组成。简介部分编制了测试目的、测试范围、定义术语、参考文档、概述等。测试用例部分逐一列示各测试用例。每个具体测试用例都将包括下列详细信息:用例编号、用例名称、测试等级、入口准则、验证步骤、期望结果(含判断标准)、出口准则、注释等。以上内容涵盖了测试用例的基本元素:测试索引、测试环境、测试输入、测试操作、预期结果、评价标准。

2) 测试用例的设置

早期的测试用例是按功能设置用例的,后来引进了路径分析法,按路径设置用例。目前演变为按功能、路径混合模式设置用例。按功能测试是最简捷的,按用例规约遍历测试每一功能。对于涉及复杂操作的程序模块,其各功能的实施是相互影响、紧密相关的,可以演变出数量繁多的变化。没有严密的逻辑分析,产生遗漏是在所难免的。路径分析是一个很好的方法,其最大的优点是可以避免漏测试。

路径分析法也有局限性,在一个非常简单的维护模块中就存在十余条路径。一个复

杂的模块有几十到上百条路径。若一个子系统有十余个或更多的模块，这些模块相互有关联，再采用路径分析法，其路径数量呈几何级增长，达5位数或更多，这样路径分析法就无法使用了。那么子系统模块间的测试路径或测试用例还是要靠传统方法来解决。这是按功能、路径混合模式设置用例的原因。

3）测试用例的设计

（1）设计基本事件的测试用例，应该参照设计规格说明书，根据关联的功能、操作，按路径分析法设计测试用例。而对孤立的功能则直接按功能设计测试用例。基本事件的测试用例应包含所有需要实现的需求功能，覆盖率达100%。

（2）设计备选事件和异常事件的测试用例，则更复杂。例如字典的代码是唯一的，不允许重复。测试需要验证：字典新增程序中已存在有关字典代码的约束，若出现代码重复必须报错，在设计编码阶段形成的文档往往对备选事件和异常事件分析描述不够详尽。而测试本身则要求验证全部非基本事件，并同时尽量发现其中的软件缺陷。

（3）可以采用等价类划分法、边界值分析法、错误推测法、因果图法、逻辑覆盖法等设计测试用例。可以根据软件的不同性质采用不同的方法。如何灵活运用各种基本方法来设计完整的测试用例，并最终实现查出隐藏的缺陷，则依靠测试设计人员的经验和水平。

（4）不论在白盒测试还是在黑盒测试中，为了节省时间和资源，提高效率，必须精心设计测试用例，也就是要从数量极大的可用测试用例中精心挑选少量的测试数据，使得采用这些测试数据能达到最佳的测试效果。

软件测试的一个致命缺陷是测试的不完全、不彻底性。由于对任何程序只能进行少量的有限测试，在发现错误时能说明程序有问题，但未发现错误时，不能说明程序中无错误。

5.3　测试执行

按照测试计划，使用测试用例对待测项目进行逐一的、详细的测试，将获得的运行结果与预期结果比较、分析和评估，判断软件是否通过了某项测试，确定开发过程中将要执行的下一步骤；同时记录、跟踪和管理软件缺陷。

在每个测试执行之后，对发现的错误都要进行相应的修改。当软件修改以后，必须运行原有的全部测试用例重新测试，并验证测试结果，这样可确保修改后的软件质量，这种测试就是回归测试。

在执行过程中，按照评价标准评价测试工作和被测软件，当发现测试工作存在问题时，应该修订测试计划，进行重测，直至测试达到规定的要求。另外，为避免在修改错误时又产生新的错误，应定期进行回归测试，即过一段时间以后，再回过头来对以前修复过的错误重新进行测试，看该错误是否会重新出现。

回归测试是确认已测试的问题已不再存在的一项工作，每进行完一个阶段应检验执行结果与测试设计文件中是否存在差异。若存在差异就应针对其进行适度的调整，可以修改测试设计文件及测试计划的进度等。

可以按照软件开发过程模型定义软件测试流程，图 5-2 所示是一个完整的测试工作流程。

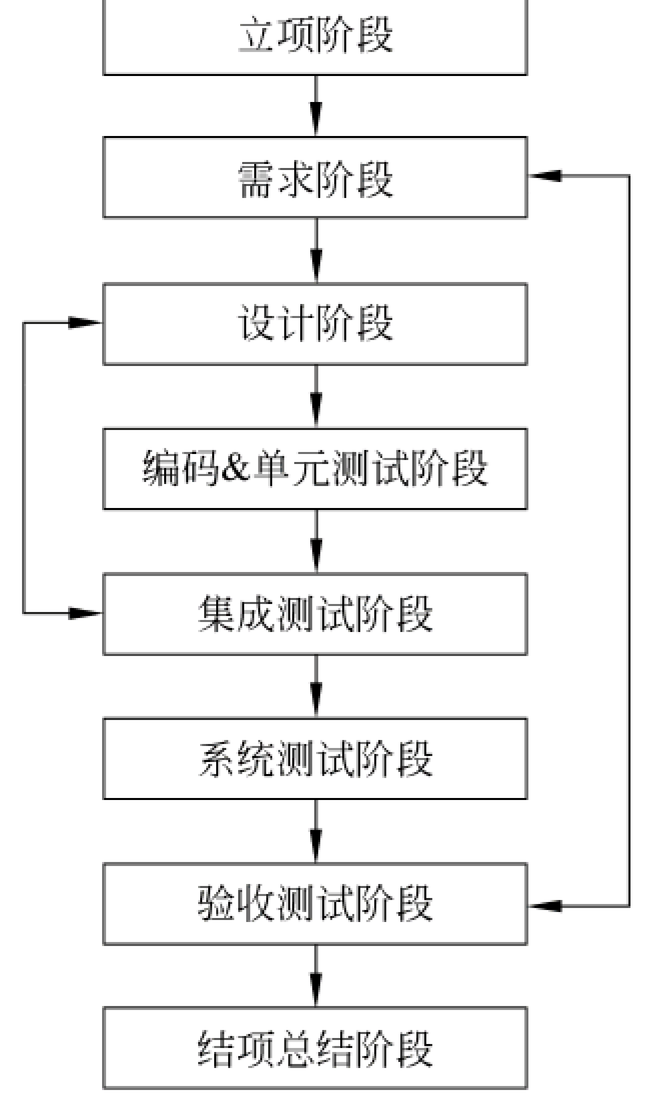


图 5-2 完整的测试工作流程

5.3.1 创建测试任务

软件测试部门的主要任务之一是按照软件测试计划、测试大纲及项目进度表，执行软件测试。为了执行软件测试，需要定义测试任务，即在某一测试阶段计划执行的测试用例的集合。

1. 使用测试任务的目的

(1) 通过将整个测试过程划分为不同的测试任务可以满足不同测试阶段的不同测试需求。

(2) 跟踪不同阶段测试用例的执行情况。

(3) 决定测试的执行状态。

(4) 同一个测试用例在不同的测试任务中会产生不同的执行报告，各自带有独立的测试执行的历史信息。

(5) 测试完成后，在任务中所有执行报告都归档。

2. 划分测试任务的标准

(1) 一个测试任务。

(2) 多个串行的测试任务。

(3) 多个并行的测试任务。

(4) 多个串行和并行的测试任务。

5.3.2 执行测试任务

按以下步骤执行测试任务。

(1) 选择测试任务中的测试用例。

(2) 执行测试用例并记录测试结果。测试结果包括：测试通过、测试失败、测试受阻。

(3) 关闭测试任务。

5.3.3 处理软件问题报告

1. 软件问题报告概念

软件测试目的就是尽可能多地发现软件问题，在软件产品发布之前，测试始终与改错过程交错进行。软件问题报告作为开发人员和测试人员协同工作的交互媒介，是测试实施过程中最重要的文档。它记录了软件问题发生的环境，如各种资源的配置情况，软件问题的再现步骤以及软件问题性质的说明，更重要的是它还记录着软件问题的处理进程。软件问题的处理进程在一定角度上反映了软件测试与开发的进程以及被测软件的质量状

况和改善过程。

软件测试人员对在测试中所发现的每一个软件问题，都要按照某种约定的标准格式写入软件问题报告。为了对所有软件问题报告进行管理，包括新建、修改、修复、验证等，需要建立一个软件问题报告管理系统，将其提供给开发和测试部门，以便他们在软件开发过程中对软件的质量进行追踪和控制。

2. 软件问题报告的内容

软件问题报告的内容主要包括以下一些。

（1）编号：是每个软件问题报告的唯一标识。

（2）作者：软件问题报告作者名称。

（3）标题：是对软件问题的简要描述。

（4）状态：软件问题报告状态。

（5）被测项目名：标识被测试的软件项目名称。

（6）被测软件版本号：标识被测试的软件版本号。

（7）软件问题严重程度：对软件问题进行分级。

（8）修改优先级：定义修改次序和时间。

（9）操作系统平台和支持软件：对发现软件问题时的软件环境进行描述，以便开发部门再现该软件问题。

（10）网络环境：对发现软件问题时的网络环境进行描述，以便开发部门再现该软件问题。

（11）软件问题再现详细步骤：对发现软件问题的步骤进行详细描述。

（12）软件问题变通和绕过方法：描述变通和绕过该软件问题的步骤。

5.4 测试总结

5.4.1 测试结果的统计

终止测试之后，就开始对测试结果进行统计和分析。可以从各种不同的角度考虑测试结果的统计。例如，依据软件错误性质及危害程度，可以把软件按错误的严重程度分为以下几类。

（1）最严重错误：如导致环境破坏，或是造成生命财产损失。

（2）非常严重的错误：如系统突然停止运作。

（3）严重错误：如系统运行不可跟踪。

（4）较严重错误：如系统结果不是所期待的。

（5）中等错误：如对系统的运行有局部的影响。

（6）较小错误：如对系统运行只有非实质性影响，如输出格式不对、显示不对。

对上述各种类型的软件错误进行统计，建立软件错误报告、分析与修改措施系统，按照相关标准的要求，制定和实施软件错误报告以及可靠性数据收集、保存、分析和处理的

规程，完整、准确地记录软件测试阶段的软件错误和收集可靠性数据。

5.4.2 测试结果的分析

对测试结果进行统计之后，就可开始对测试结果进行分析。简单地说，分析过程就是一个错误与软件要求的功能相匹配的过程。大体上可从以下几点对测试结果进行分析。

1. 能力

描述了经测试证实了的软件的能力。如果所进行的测试是为了验证一项或几项特定性能要求的实现，应提供这方面的测试结果与要求之间的比较，并确定测试环境甚至与实际运行环境之间可能存在的差异对能力的测试所带来的影响。

2. 缺陷和限制

描述了经测试证实的软件缺陷和限制，说明每项缺陷和限制对软件性能的影响，并说明全部测得的性能缺陷的累积影响和总影响。

3. 建议

对每项缺陷提出改进建议如下。

(1) 各项修改可采用的修改方法。

(2) 各项修改的紧迫程度。

(3) 各项修改预计的工作量。

(4) 各项修改的负责人。

4. 评价

说明该项软件的开发是否已达到预定目标，能否交付使用。

5.4.3 测试报告的编写

测试活动结束后必须编写软件可靠性测试报告，对测试项及测试结果在测试报告中加以总结归纳。测试报告应具备下列内容。

(1) 产品标识。

(2) 使用的配置(硬件和软件)。

(3) 使用的文档。

(4) 产品说明、用户文档、程序和数据的测试结果。

(5) 与需求不相符的功能项列表。

(6) 测试的最终日期。

这种规范化的过程管理控制有利于获得真实有效的数据，为最终得到客观的评估结果奠定基础。测试报告的编写凝聚着本次测试所有的工作，也是对整个工作的一次认可和总结。

小 结

本章系统地介绍了测试的整个过程。这个过程包括：测试计划、测试设计、测试执行和测试总结等，并对每一个要点都做了详细的介绍。作为一项工程，成功的测试需要完成的不仅仅是上述的工作，还有许多工作。

习 题 5

1. 编写软件测试计划的 3 个目的是________，________和________。
2. 下列哪些不是软件测试策略的特征？(　　)
   A. 测试开始于单元级，然后延伸到整个系统中
   B. 不同的测试技术适用于不同的时间点
   C. 测试是由软件的独立测试组织来管理的
   D. 测试和调试是相同的活动，且调试必须能够适应任何的测试策略
3. 为了保证测试的质量，将测试过程分成 5 个阶段，即：________，________，________，________，________。
4. 定义软件测试用例的六元组是________，________，________，________，________和________。
5. 对①防范错误发生；②如何处理错误；③如何告知错误 3 种情况，分别设计 3 个错误处理测试用例。并说明测试原因和所对应的测试用例。
6. 划分测试任务的标准是什么？
7. 软件错误的严重程度可以分为几种？分别是什么？
8. 下列哪些是测试人员应具备的素质？(　　)
   A. 沟通能力　B. 技术能力　C. 洞察力　D. 共同价值观
   E. 自信　F. 交流　G. 记忆　H. 怀疑
   I. 耐心　J. 自我激励
9. 试论述测试人员应该如何与程序员进行交流。

第 6 章　Web 应用测试

学习要点：

❖ Web 应用设计测试。

❖ Web 应用开发测试。

❖ Web 应用运行测试。

❖ J2EE 平台的测试。

❖ .NET 平台的测试。

6.1　Web 测试概述

随着 Web 应用系统广泛应用，针对其功能和性能的测试要求也越来越高，然而由于 Web 应用程序融合了 HTML、Java、JavaScript、VBScript 等大量的新技术，同时它还依赖于链接、数据库、网络等很多其他的因素，使得 Web 应用系统的测试不同于传统软件的测试，测试变得更为复杂。

6.1.1　Web 系统的结构

Web 系统以 Browser/Server（浏览器/服务器）的访问为主，包含客户端浏览器、Web 应用服务器、数据库服务器的软件系统。在逻辑上，B/S 结构是一种多层构架结构，分有界面层、业务逻辑层和数据层。在结构上，分有客户端部分、传输网络部分和服务器端部分。

1. 应用服务器的分类

应用服务器基于其结构和应用领域的不同而分类。基于应用领域和范围的不同，应用服务器分为如下几类。

（1）Web 服务器。通过 MS IIS、BEA WebLogic 和 Apache 等中间件、插件来提供 Internet/Intranet 的 Web 服务，实现与客户之间的数据共享与交换。

（2）数据库服务器。其主要功能是提供数据库查询、处理的平台，通过 Oracle/SQL Server/Informix/DB2 等大中型的数据库管理系统来构建。

（3）实时通信服务器。提供数据实时通信、消息传递等服务，如实时通信平台 MSN、OICQ 等专用服务器。

(4) 邮件管理服务。实现邮件管理的服务器系统,如 MS Exchange Server。

(5) 群件服务器。提供工作群组之间的协作服务,如 Lotus Domino 平台。

(6) 文件/打印服务器。实现文件共享和打印功能的服务器系统。

(7) 构件的服务器。指以 Java 技术为基础 J2EE 构件的服务。

2. C/S 和 B/S 结构

C/S 结构是目前常用的应用服务模式之一,它基于客户/服务器范型进行工作。在服务器端,一般采用高性能的 PC、工作站或者专业服务器,并根据需要采用大型的数据库系统,如 Oracle、Sybase、Informix 或者 MS SQL Server;而客户端则需要安装专用的客户端软件。C/S 结构是一种常用的结构,例如客户端基于 Outlook Express,服务器端基于 Outlook Exchange Server,又如 MSN、网络游戏等都是一些典型的 C/S 结构的应用。

C/S 结构能充分发挥客户端的处理功能,将很多部分的工作,如计算、数据采集等通过客户端处理以后提交给服务器,这样减少了服务器的压力,从而能很快响应客户端的需求。由于 C/S 的客户端需要安装用户专用的客户端软件,这样给开发、安装、升级、维护以及数据存储都带来一系列的问题。同时,客户端程序受到操作系统的限制,如果应用程序不支持平台特性,只能运行在 Windows 系统上,就无法在 Linux、Solaris 或者其他平台运行。为了解决这一问题,出现了 B/S 结构。

B/S 结构克服了 C/S 需要在客户端机器上安装程序及维护不方便的缺点,不需要额外的客户端程序支持,而是通过浏览器与服务器进行通信和数据传输,由此 B/S 结构更容易维护和升级。目前,操作系统都自带浏览器,包括 Windows、Mac OS、UNIX、Linux 等平台都已安装浏览器,最常见的浏览器有 Microsoft 的 Internet Explorer、Netscape 的 NS 等。

3. 三层和多层结构

数据处理复杂性越来越高,而且客户需求日益增长,传统的双层 C/S 结构或 B/S 结构难以实现上述要求。因此,出现了三层或者多层结构的概念。

三层结构主要是将应用层分离出 3 个相互隔离的逻辑层,每一层都定义好一套接口集。第一层是表示层,主要由类似于图形用户界面的部分组成;中间层为业务层,由应用逻辑和业务逻辑构成;而第三层为数据层,包括了应用程序中所需要的数据。

用户通过表示层调用中间层(应用逻辑)的代码来获取需要的数据,表示层接收数据并且按照适当的格式显示出来。从用户界面中分离出应用逻辑,极大地增强了应用程序设计的灵活性。在应用逻辑层对表示层提供了一套定义清晰的接口的情况下,适应不同的信息源。数据层可以是 Oracle、Informix 这样的数据库,也可以是 XML 文档集,还可以是 LDAP 服务器的目录服务。

三层结构的出现对应用层次划分还没有终止,最终目的是为了创建多层体系结构,在多层体系结构中,应用逻辑的划分是根据功能来进行的。多层结构的划分如下。

(1) 用户接口层:负责处理用户与应用程序之间的交互过程。它可以是一个通过防火墙的 Web 浏览器,也可以是一般的桌面应用程序,甚至是无线设备或其他设备。

(2) 表示逻辑层:定义了用户界面要显示的内容和如何处理用户的需求。对于相应

的客户，表示逻辑层可以有不同的版本。

(3) 业务逻辑层：通过与应用数据的通信，对应用的业务规则实现建模。

(4) 基础框架服务层：提供了应用系统需要的其他功能，如消息传输。

(5) 数据层：存放数据的数据库。

采用这种结构的目的是为了把数据和表示数据的部分分离出来，它们之间通过应用/业务逻辑来控制信息的流向。J2EE 服务器架构是三层结构和多层结构的经典例子。

6.1.2 Web 测试目的与计划

1. Web 测试的目的

测试的目的是为了寻找软件中的错误，例如测试人员给某网站做测试，并递交了一份简单的测试报告："当 100 个用户共同单击某提交按钮时，发生提交失败"。对于测试人员来说，他已经完成了他自己的任务，找出了错误，但是，报告中并未提及造成提交失败的原因，是硬件资源不足、网络问题、支撑软件参数设置错误还是应用开发问题等。测试的目的是证伪，但不能片面地理解为简单地再也找不到错误就可以了。软件测试应该经历以下 4 个步骤。

(1) 测试人员描述发现的问题。

(2) 测试人员详细阐明是在何种情况下测试发现的问题，包括测试的环境、输入的数据、发现问题的类型、问题的严重程度等情况。

(3) 测试人员协同开发人员一起去分析错误的原因，找出软件的问题所在。

(4) 测试人员根据解决的情况进行分类汇总，以便日后进行软件设计的时候提供参考，避免以后出现类似软件缺陷。

2. Web 测试的计划

制定 Web 测试计划针对一个 Web 应用程序进行测试，需要制定详细的测试计划，计划的内容归纳为以下几点。

(1) 首先对被测的 Web 应用程序进行需求分析，对所做的测试做一个简要的介绍，包括描述测试的目标和范围，测试的目标要指明实现什么样的功能，总结基本文档和主要活动等。

(2) 写出测试策略和方法，主要包括测试开始的条件、测试的类型、测试开始的标准以及所测试的功能、测试通过或失败的标准、结束测试的条件、测试过程中遇到什么样的情况终止和怎么处理后恢复等。

(3) 确定测试环境的要求，包括软件和硬件方面，选择合适的测试工具。

(4) 主要针对测试的行为，描述测试的细节，包括测试用例列表、进度表和错误等级分析，对测试计划的总结和在测试过程中会出现的风险分析等。

6.1.3 Web 系统的测试策略

基于系统架构考虑，Web 系统的测试可以分为客户端测试、服务器端测试、网络上测试。基于职能划分，可分为应用功能测试、Web 应用服务测试、安全系统测试、兼容性测

试和易用性测试。基于开发阶段来划分，可分为设计的测试、编码的测试和系统的测试。Web 软件的开发同样要经过需求分析、设计、编码和实施阶段，所以测试存在于软件开发全过程。Web 应用系统的测试可以分为 Web 应用设计测试、Web 应用开发测试和 Web 应用运行测试。

6.2　Web 应用设计测试

在 Web 应用系统的设计阶段，测试的主要内容就是对 Web 设计进行全面性、标准性、适合性检查，根据 Web 应用系统的架构，将 Web 设计的测试分为总体架构设计的测试、客户端设计的测试和服务器端设计的测试 3 个部分。

6.2.1　总体架构设计的测试

在 Web 应用系统中，服务器端涉及各种类型的硬件、操作系统、服务进程、服务器和数据库等软件的组合。上述问题是 Web 应用系统的关键问题。对总体架构设计的测试可从下述方面考虑。

1. 客户端

在瘦客户端系统中，客户端仅作少量的处理，业务逻辑规则多数在服务器端执行。这种模式适于用户数量巨大、用户分散的应用系统，门户网站、新闻站点等均为瘦客户端 Web 系统。胖客户端即运行应用程序的用户界面，又执行部分业务逻辑。这种模式适合交互操作频繁、业务复杂和安全性高的应用系统。胖客户端能够减轻服务器负担，但需要在客户端安装一些应用程序。测试的任务是验证设计中采用的模式能否满足系统需求。

2. Web 架构

在确定服务器端的组成时，要从成本、功能、性能、安全性传输实时性等方面考虑，设计服务器端的服务器群。对 Web 架构设计的测试要验证 Web 架构是否满足上述要求，各组成部分是否兼容。

3. 服务器的配置和分布

服务器软件可以分布在多个物理服务器上，Web 架构设计的测试要验证服务器的配置和分布是否满足用户的要求。

6.2.2　客户端设计的测试

客户端设计包括功能设置、信息组织结构和页面设计等。

1. 功能设置测试

功能设置以企业内部工作需要为主，常用的功能如下。

(1) 信息服务。信息服务为用户提供多种动态和静态信息。

(2) 办公自动化。办公自动化是指提供企业内部通信、工作流控制等功能，进而实现企业办公自动化。

(3) Internet。接入 Internet，获得 Internet 提供的各种服务。

客户端设计测试主要是测试功能设置是否满足用户需求。

2. 信息组织结构设计的测试

信息组织结构设计主要有线性结构设计、层次结构设计和非线性结构设计。

(1) 线性结构设计。线性结构的信息按顺序链接，允许用户向前或向后检索信息内容，不允许用户以非顺序方式浏览信息内容。

(2) 非线性结构设计。非线性结构没有明显的结构特征，允许用户可以随意在信息中漫步，完全自由地改变信息浏览路径。

(3) 层次结构设计。层次结构以树型方式组织信息，具有层次清楚、易于查找等特点。

信息组织结构设计的测试主要是验证设计是否符合信息的特点，能否使用户直观而方便地浏览所需的信息。

3. 页面设计测试

好的页面信息层次清楚、精致美观、可理解性强。对页面设计测试主要考虑如下方面。

(1) 一致性强。

(2) 界面友好、导航直观。

(3) 建立了页面文件命名体系。

(4) 页面布局合理。

(5) 适合多种浏览器。

6.2.3 服务器端设计的测试

服务器端设计的测试主要包括容量规划的测试、安全系统设计的测试和数据库设计的测试。通过这些内容的测试，可以提高 Web 应用系统的性能、安全性、可靠性等。

1. 容量规划的测试

容量规划与 Web 应用系统的性能密切相关，对容量规划测试异常重要。评价容量规划设计的重点如下。

(1) 估算点击率是否满足需求。

(2) 估算延迟和流量是否满足需求。

(3) 估算 Web 应用系统所需服务器资源消耗。

2. 安全系统设计的测试

(1) 审核采用的加密方式能否满足用户要求。

(2) 评估设计中采取了的常识性安全策略的有效性。

(3) 审核采用的防火墙和安全级别是否满足用户需求。

(4) 验证网络的防毒体系设计是否全面、多层次和全方位。

3. 数据库设计的测试

除了数据库的常规测试之外，主要进行应用程序的数据处理能力的测试，即获得应用

程序的数据处理能力的上限。

6.3 Web应用开发测试

Web应用系统开发测试主要指在Web应用的开发阶段，对Web应用的源代码和组件进行测试，进而保证代码的正确性。

1. 代码测试

对Web应用系统的代码测试主要指源代码规则分析、链接测试和页面静态对象测试等。

1）源代码规则分析

使用检查工具，将源代码与编码标准语言规则相对照，进而找出两者之间的不一致性和源代码的潜在错误。

2）链接测试

链接测试主要包括以下3个方面。

（1）测试链接是否按指示那样成功。

（2）测试链接的页面是否存在。

（3）测试系统中无孤立的页面。

具体的测试方法是逐一检查链接的有效性、可达性和正确性。

3）框架测试

（1）检查框架是否随浏览窗口的变化而自动调整大小。

（2）检查是否需要提供滚动条。

（3）检查能否在目标框架中打开新页面。

2. 组件测试

Web组件是一个软件单元，被用于Web系统中，用户的使用请求可以通过浏览器的解释传递给组件，组件的执行结果经过浏览器传递给用户。组件测试分为静态测试和动态测试两种。主要有以下测试。

（1）表单测试。

（2）脚本测试。

（3）ASP测试。

（4）Cookies测试。

（5）CGI测试。

（6）ActiveX控件测试。

6.4 Web应用运行测试

Web应用设计测试通过之后，进入了运行测试，其主要内容包括功能、性能、易用性、兼容性和安全性测试等。测试方法为根据功能说明书、需求说明书等文档，设计测试用

例，然后进行测试。测试手段是人工测试、工具测试和评估等。

1. 功能性测试

1）功能测试

在这里，功能测试是指Web应用系统的基本功能的测试，主要包括客户端的选择、客户端浏览器的配置和内容测试，内容测试可以用来检测Web应用系统提供信息的正确性、准确性和相关性等。

（1）链接测试。链接是Web应用系统的一个主要特征，主要功能是在页面之间切换和指导用户去一些不知道地址的页面的主要手段。链接测试可分为如下3个方面。

- 测试所有链接是否按指示的那样链接到了该链接的页面。
- 测试所链接的页面是否存在。
- 保证Web应用系统中没有孤立的页面，孤立页面是指没有链接指向该页面。

链接测试必须在集成测试阶段完成，也就是说，在整个Web应用系统的所有页面开发完成之后进行链接测试。链接测试可以自动进行，现在已经有许多工具可以采用。

（2）表单测试。当用户给Web应用系统管理员提交信息时，就需要使用表单操作，例如用户注册、登录、信息提交等。在这种情况下，必须测试提交操作的完整性，以校验提交给服务器的信息的正确性。例如用户填写的出生日期与职业是否恰当，填写的所属省份与所在城市是否匹配等。如果使用了默认值，还要检验默认值的正确性。如果表单只能接收指定的某些值，则也要进行测试。例如只能接收某些字符，测试时可跳过这些字符，观看系统是否会报错。

（3）Cookies测试。Cookies通常用来存储用户信息和用户在某应用系统的操作，当一个用户使用Cookies访问了某一个应用系统时，Web服务器将发送关于用户的信息，把该信息以Cookies的形式存储在客户端计算机上，这可用来创建动态和自定义页面或者存储登录等信息。如果Web应用系统使用了Cookies，就必须检查Cookies是否能正常工作。测试的内容可包括Cookies是否起作用、是否按预定的时间进行保存、刷新对Cookies有什么影响等。

（4）设计语言测试。Web设计语言版本的差异可以引起客户端或服务器端的问题，例如使用哪种版本的HTML等。当在分布式环境中开发时，开发人员都不在一处，这个问题就显得尤为重要。除了HTML的版本问题外，不同的脚本语言，例如JavaScript、ActiveX、VBScript或Perl等也要进行验证。

（5）数据库测试。在Web应用技术中，数据库起着重要的作用，数据库为Web应用系统的管理、运行、查询和实现用户对数据存储的请求等提供空间。在Web应用中，最常用的数据库类型是关系型数据库，可以使用SQL对信息进行处理。在使用了数据库的Web应用系统中，一般情况下可能发生两种错误，分别是数据一致性错误和输出错误。数据一致性错误主要是由用户提交的表单信息不正确而造成的，而输出错误主要是由网络速度或程序设计问题等引起的，针对这两种情况，可分别进行测试。

2）性能测试

（1）连接速度测试。连接速度测试是测试Web系统的连接速度。用户可以通过电话拨号或宽带上网，与Web系统连接的速度与上网方式有关，当下载一个程序时，用户需

要等较长的时间,但如果仅仅访问一个页面就很短暂。另外,有些页面有超时的限制,如果响应速度太慢,用户可能还没来得及浏览内容,就需要重新登录了。而且连接速度太慢,还可能引起数据丢失,使用户得不到真实的页面。因此,连接速度测试不可缺少。

(2) 负载测试。负载测试是为了测量 Web 系统的负载能力,以保证 Web 系统在需求范围内能正常工作。负载能力可以是某个时刻同时访问 Web 系统的用户数量,也可以是在线数据处理的数量。例如:Web 应用系统能允许用户同时在线的数量,如果超过了这个数量,将出现什么现象,Web 应用系统能否处理大量用户对同一个页面的请求等问题。

(3) 压力测试。压力测试是指实际破坏一个 Web 应用系统的反映。压力测试是测试 Web 应用系统的限制和故障恢复能力,也就是测试 Web 应用系统在什么情况下会崩溃。压力测试的区域包括表单、登录和信息传输页面等。压力测试是在 Web 系统发布以后在实际的网络环境中进行的测试。因为一个企业内部员工,特别是项目组人员总是有限的,而一个 Web 系统能同时处理的请求数量将远远超出这个限度,所以只有放在 Internet 上接受负载测试,其结果才是正确的。

3) 用户界面测试

(1) 导航测试。导航一方面描述了用户在一个页面内操作的方式,另一方面描述了在不同的链接页面之间是否合理链接,导航测试主要包括下列问题。

- 导航是否直观。
- Web 系统的主要部分是否可通过主页存取。
- Web 系统是否需要站点地图。
- 搜索引擎或其他的导航帮助等。
- 导航、菜单、链接的风格是否一致。

(2) 图形测试。在 Web 应用系统中,适当的图片和动画既能起到广告宣传的作用,又能起到美化页面的功能。一个 Web 应用系统的图形可以包括图片、动画、边框、颜色、字体、背景、按钮等。图形测试的主要内容如下。

- 图形用途是否明确。图片或动画要有规则地堆放,以免浪费传输时间。Web 应用系统的图片要能清楚地表示某件事情,一般都链接到某个具体的页面。
- 字体的风格是否一致。
- 背景颜色与字体颜色、前景颜色是否相搭配。
- 图片的大小和质量是否得当,最好使图片的大小减小到 30KB 以下。
- 需要验证的文字回绕是否正确。如果说明文字指向右边的图片,应该确保该图片出现在右边。不要因为使用图片而使窗口和段落排列古怪。

通常来说,使用少量或尽量不使用背景是个较好的选择。如果想用背景,那么最好使用单色的,并和导航条一起放在页面的左边。

(3) 内容测试。内容测试用于检验 Web 应用系统提供信息的正确性、准确性和相关性。信息的正确性是指信息是可靠的还是误传的。例如,在商品价格单中,错误的价格可能引起财政问题甚至导致法律纠纷;信息的准确性是指是否有语法或拼写错误。这种测试通常使用一些文字处理软件来进行,例如使用 Microsoft Word 的“拼音与语法检查”功

能;信息的相关性是指是否在当前页面可以找到与当前浏览信息相关的信息列表或入口,也就是一般 Web 站点中的相关文章列表。

对于开发人员来说,先有功能然后才对这个功能进行描述。先确定功能,然后开始开发,在开发的时候,开发人员可能不注重文字表达,他们添加文字可能只是为了对齐页面,这样出来的产品可能产生严重的误解。因此测试人员和公关部门一起检查内容的文字表达是否恰当。否则,可能陷入麻烦之中,也可能引起法律方面的问题。测试人员应确保站点看起来更专业。过分地使用粗体字、大字体和下划线会让用户感到不舒服。在进行用户可用性方面的测试时,最好先由图形设计专家对站点进行评估。最后,需要确定是否列出了相关站点的链接。很多站点希望用户将邮件发到一个特定的地址,或者从某个站点下载浏览器。

(4) 表格测试。需要验证表格是否设置正确。用户是否需要向右滚动页面才能看见产品的价格,把价格放在左边,而把产品细节放在右边是否更有效,每一栏的宽度是否足够宽,表格里的文字是否都有折行,是否有因为某一格的内容太多,而将整行的内容拉长等。

(5) 整体界面测试。整体界面是指整个 Web 应用系统的页面结构设计,是给用户的一个整体感。例如:当用户浏览 Web 应用系统时是否感到舒适,是否凭直觉就知道要找的信息在什么地方,整个 Web 应用系统的设计风格是否一致。对整体界面的测试过程其实是一个对最终用户进行调查的过程。一般 Web 应用系统采取在主页上做一个调查问卷的形式来得到最终用户的反馈信息。

对所有的用户界面测试来说,都需要有外部人员(与 Web 应用系统开发没有联系或联系很少的人员)的参与,最好是最终用户的参与。

4) 兼容性测试

(1) 平台测试。常见的操作系统有 Windows、UNIX、Macintosh、Linux 等。Web 应用系统的最终用户究竟使用哪一种操作系统,取决于用户系统的配置。这就可能会发生兼容性问题,同一个应用可能在某些操作系统下能正常运行,但在另外的操作系统下可能会运行失败。因此,在 Web 系统发布之前,需要在各种操作系统下对 Web 系统进行兼容性测试。

(2) 浏览器测试。浏览器是 Web 客户端最核心的构件,来自不同厂商的浏览器对 Java、JavaScript、ActiveX、plug-ins 或不同的 HTML 规格有不同的支持。例如,ActiveX 是 Microsoft 的产品,是为 Internet Explorer 而设计的;JavaScript 是 Netscape 的产品;Java 是 Sun 的产品;等等。另外,框架和层次结构风格在不同的浏览器中也有不同的显示,甚至根本不显示。不同的浏览器对安全性和 Java 的设置也不一样。测试浏览器兼容性的一个方法是创建一个兼容性矩阵。在这个矩阵中,测试不同厂商、不同版本的浏览器对某些构件和设置的适应性。

(3) 分辨率测试。分辨率测试是指页面版式在 640×400、600×800 或 1024×768 的分辨率模式下是否显示正常,字体是否太小以至于无法浏览,或者是否太大,文本和图片是否对齐等内容。

(4) Modem/连接速率测试。用户在下载文章或演示的时候,可能会等待比较长的

时间,但却不会耐心等待首页的出现。Modem/连接速率测试下载一个页面需要多长时间。

(5) 打印机测试。用户可能会将网页打印下来,因此网页在设计的时候要考虑到打印问题,需要验证网页打印是否正常。有时在屏幕上显示的图片和文本的对齐方式可能与打印出来的东西不一样。测试人员至少需要验证订单确认页面打印正常。

5) 安全性测试

即使站点不接受信用卡支付,安全问题也是非常重要的。Web 站点收集的用户资料只能在公司内部使用。如果用户信息被黑客泄露,客户在进行交易时就不会有安全感。

(1) 目录设置。Web 安全的第一步就是正确设置目录。每个目录下应该有 index. html 或 main. html 页面,这样就不会显示该目录下的所有内容。

(2) SSL。很多站点使用 SSL 进行安全传送。进入一个 SSL 站点是因为浏览器出现了警告消息,而且在地址栏中的 HTTP 变成 HTTPS。如果开发部门使用了 SSL,测试人员需要确定是否有相应的替代页面(适用于 3.0 以下版本的浏览器,这些浏览器不支持 SSL)。当用户进入或离开安全站点的时候,请确认是否有相应的提示信息;是否有连接时间限制,超过限制时间后出现什么情况等。

(3) 登录。有些站点需要用户进行登录,以验证他们的身份。这样对用户是方便的,他们不需要每次都输入个人资料。需要验证系统阻止非法的用户名/口令登录,而能够通过有效登录。用户登录是否有次数限制,是否限制从某些 IP 地址登录,如果允许登录失败的次数为 3,在第三次登录的时候输入正确的用户名和口令,就能通过验证;口令选择是否有规则限制,是否可以不登录而直接浏览某个页面;Web 应用系统是否有超时的限制,也就是说,用户登录后在一定时间内(例如 15 分钟)没有点击任何页面,是否需要重新登录才能正常使用。

(4) 日志文件。在后台,要注意验证服务器日志工作正常。日志是否记录所有的事务处理?是否记录失败的注册企图?是否记录被盗信用卡的使用?是否在每次事务完成的时候都进行保存?记录 IP 地址吗?记录用户名吗?

(5) 脚本语言。脚本语言是常见的安全隐患,每种语言的细节有所不同。有些脚本允许访问根目录,其他只允许访问邮件服务器。但是经验丰富的黑客可以将服务器用户名和口令发送给他们自己,找出站点使用了哪些脚本语言,并研究该语言的缺陷。还需要测试没有经过授权,就不能在服务器端放置和编辑脚本的问题。最好的办法是订阅一个讨论站点使用的脚本语言安全性的新闻组。

6) 接口测试

在很多情况下,Web 站点不是孤立的。Web 站点可能会与外部服务器通信,请求数据、验证数据或提交订单。

(1) 服务器接口。第一个需要测试的接口是浏览器与服务器的接口。测试人员提交事务,然后查看服务器记录,并验证在浏览器上看到的正好是服务器上发生的。测试人员还可以查询数据库,确认事务数据已正确保存。

这种测试可以归到功能测试中的表单测试和数据校验测试中。

(2) 外部接口。有些 Web 系统有外部接口。例如,网上商店可能要实时验证信用卡数据以减少欺诈行为的发生。测试的时候,要使用 Web 接口发送一些事务数据,分别对有效信用卡、无效信用卡和被盗信用卡进行验证。如果商店只使用 Visa 卡和 Mastercard 卡,可以尝试使用 Discover 卡的数据(简单的客户端脚本能够在提交事务之前对代码进行识别,例如 3 表示 American Express,4 表示 Visa,5 表示 Mastercard,6 表示 Discover)。通常,测试人员需要确认软件能够处理外部服务器返回的所有可能的消息。

(3) 错误处理。最容易被测试人员忽略的地方是接口错误处理。通常试图确认系统能够处理所有错误,但却无法预测系统所有可能的错误。尝试在处理过程中中断事务,看看会发生什么情况?订单是否完成?尝试中断用户到服务器的网络连接。尝试中断 Web 服务器到信用卡验证服务器的连接。在这些情况下,系统能否正确处理这些错误?是否已对信用卡进行收费?如果用户自己中断事务处理,在订单已保存而用户没有返回网站确认的时候,需要由客户代表致电用户进行订单确认。

2. 易用性测试

易用性测试主要指检测 Web 应用系统是否易于使用。易用性是指对于用户来说,Web 应用系统易于使用的程度。主要从下述 3 个方面进行测试。

(1) 界面测试。

(2) 辅助功能测试。

(3) 图形测试。

3. 负载压力测试

(1) 负载测试。负载测试是测量 Web 应用系统在某一负载级别上的性能,用以保证 Web 应用系统在需求范围内能正常工作。例如,负载级别可以是某时刻同时访问 Web 应用系统的用户数量。

(2) 压力测试。压力测试是指破坏一个 Web 应用系统,测试系统的反映。压力测试是测试系统的限制能力和故障恢复能力。也就是说,测试 Web 应用系统在什么情况下会崩溃。Web 应用系统的压力测试步骤如下。

① 确定交易执行响应时间。

② 估计 Web 应用系统能够承受的最大并发用户数量。

③ 模拟用户要求,从一个较小的负载开始,逐渐增加模拟用户的数量,直到系统不承受为止。

④ 如果负载没有达到要求,那么应该优化这个 Web 程序。

4. 客户端配置与兼容性测试

客户端配置与兼容性测试的目的是发现 Web 应用系统在可能的用户环境下运行程序时出现的错误。由于 Web 应用系统采用的是浏览器/服务器模式,故将测试的重点放在客户端,可以分为下述 3 个方面。

(1) 浏览器配置测试。

(2) 平台兼容性测试。

(3) 浏览器兼容性测试。

6.5 Web服务器测试

服务器是具有很强的集中处理能力、可靠性、可扩展性和可管理性的通用计算设备。随着网络的不断发展，其应用范围越来越广，除了一些专用的服务器，如网关服务器、代理服务器等之外，大多数服务器都被用来提供业务或商业应用的计算服务，成为基于网络的应用系统的核心。

应用服务器用于执行单一的或者是专用的功能，其中常用的应用服务器有 Web 服务器、数据库服务器、FTP 服务器、邮件服务器、文件共享服务器等。随着 Internet 的广泛应用，Web 的应用越来越广泛，基于 Web 服务器的应用系统也变得非常普及，因此，对 Web 的要求也越来越高。在 Web 服务器测试中，一般考虑如下内容。

(1) 功能测试：主要测试页面功能和逻辑功能是否满足功能规格设计说明书的需求，包括基本的逻辑、页面特性等。

(2) 用户界面测试：包括用户页面是否和设计保持一致，一些基本的页面元素（如按钮、表单、图片、文字等）的尺寸大小、边距、间距、布局是否和功能规格说明书相符，文字是否有错误拼写等。

(3) 常用 Web 元素测试：包括页面的链接、图片/文字信息、表单处理、脚本语言错误校验等。

(4) 负载/压力测试：检验 Web 程序的负载效率，当有较大的客户访问量出现时，需要对页面的执行效率进行模拟，以实现整个系统处理交易的能力，这也是为了保证系统的稳定性。在负载/压力测试过程中，一般需要一些辅助的测试工具进行模拟测试。

(5) 安全性测试：包括页面、Web 服务器是否存在安全性漏洞、加密信息的保密性、防止黑客攻击能力等。

(6) 兼容性测试：各种操作系统（Windows、Mac OS、Solaris、Linux）与不同的浏览器以及浏览器版本的组合（Internet Explorer、Netscape 不同的版本）是否能正常执行，在做兼容性测试时，需要对一些已知的因素进行测试，例如 Netscape 浏览器本身的一些缺陷、Mac OS 的限制等。

(7) 网络链接测试：包括不同的网络链接方式的速度、效率，同时需要考虑是否需要代理服务器，在代理服务器上怎样执行 Internet 链接等。

(8) 其他方面的测试：如系统的不同分辨率下的页面显示、流量测试等。

6.5.1 Web 元素功能测试

Web 的常用元素主要包括图片、文字、HTML 语言、脚本语言、表单等，由这些元素构成超级链接。在基于 Web 的应用中，对网页的功能测试主要包括以下几个方面。

1. 页面链接测试

页面的链接是使用户从一个页面浏览到另外一个页面的重要手段，在做页面链接测试时，需要验证两个问题。

(1) 该页面是否存在，如页面不能显示信息，则视为页面链接无效。引起链接页面无效的因素有很多种，主要有页面文件在 Web Server 不存在及链接地址不正确等。

(2) 该页面是否跳转到所规定的页面，主要是验证页面正确性，这种测试也应该在 Web 功能测试部分被考虑。

2. 设计语言测试

设计语言主要是指 HTML 语言和不同的脚本语言，在某些情况下，HTML 语言随着客户浏览器的不同可能会产生不同的效果，因此，这也是测试中需要考虑的因素。

3. Web 图形测试

Web 图形是一种常见的显示信息的手段，如 GIF 图片、Flash 等，由于图形和文本经常混合在一起使用，因此，在 Web 图形测试的时候，不仅要确认文本是否正确，同时需要确认图片的内容和显示，如文字是否正确地环绕图片，图片的文字提示是否正确，图片所指向的链接是否正确等。当然，页面的负载测试中，图片显示也是一个重要因素，某些时候，在网络状态不好且图片文件比较大的时候，可能会遇到链接超时的错误，这些也需要被考虑在图形测试之内。图形测试还应当考虑显示问题，例如不同分辨率下的图形显示是否正确，需要浏览器附加程序支持的图形(如 Flash 动画)能否正确加载等。

4. 表单测试

表单在访问者和服务器之间建立了一个对话，允许使用文本框、单选按钮和选择菜单来获取信息，而不是用文本、图片来发送信息。通常情况下，要处理从站点访问者发来的响应(即表单结果)，需要使用某种运行在 Web 服务器端的脚本(如 PHP、JSP)，同时在提交访问者输入表单的信息之前也可能需要用浏览器运行在客户端的脚本。在进行表单测试时，需要保证应用程序能正确处理这些表单信息，并且后台程序能够正确解释和使用这些信息。举个例子，用户可以通过表单提交来实现联机注册。当注册完之后，应该从 Web 服务器上返回注册成功的消息，处理过程如图 6-1 所示。

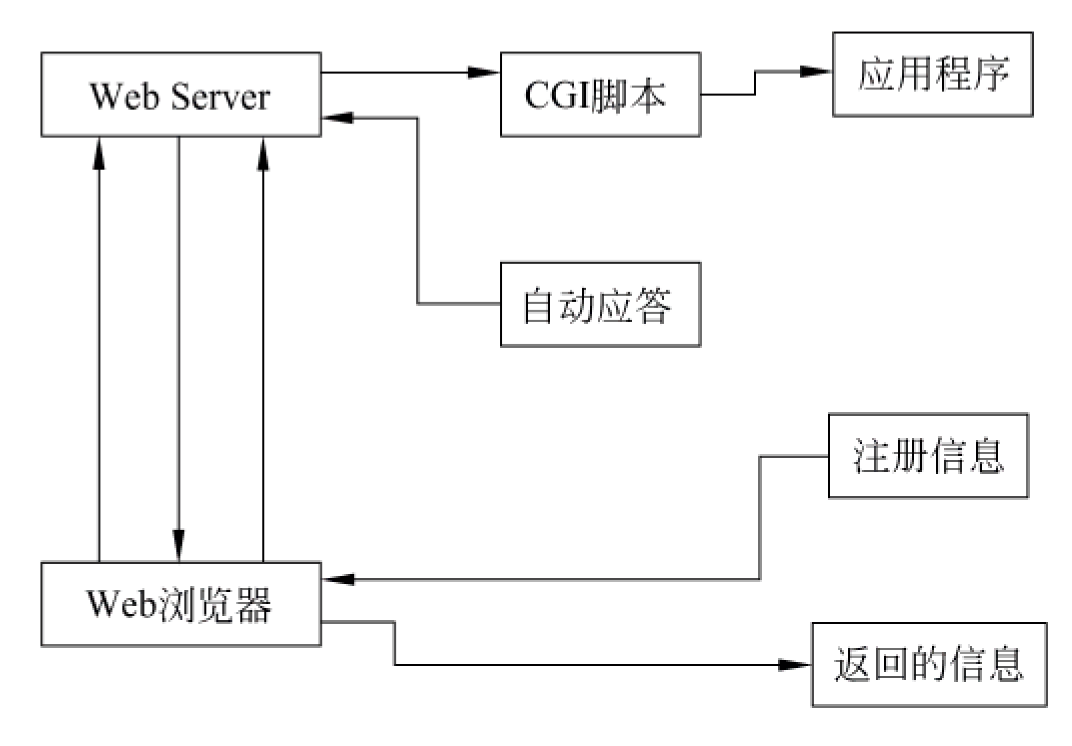


图 6-1 联机注册程序处理过程

其过程如下。

(1) Web 服务器将表单传送到 Web 浏览器。

(2) Web 浏览器显示需要访问者填写的注册信息表单。

(3) 访问者将提交按钮和数据传送到 Web 服务器。
(4) Web 服务器将表单数据传送到 CGI 脚本。
(5) 处理表单结果的 CGI 脚本将数据格式化并将其发送给相应应用程序处理。
(6) CGI 脚本产生一个验证消息并将其传送给 Web 服务器。
(7) Web 服务器将验证消息发送给 Web 浏览器,以便显示。

6.5.2 Web 安全性测试

随着 Internet 的普及,网上购物、网上交易、电子银行等新的信息交易方式走进人们的生活,网络安全越来越不容忽视。在这些应用中,通常要使用 Web 页面来传送一些重要的信息,如信用卡信息、用户资料信息等,一旦这些信息被黑客捕获,后果不堪设想。

在 Web 的安全性测试中,通常需要考虑下列的情形。

(1) 数据加密:某些数据需要进行加密和过滤后才能进行数据传输,例如用户信用卡信息、用户登录密码信息等。此时需要进行相应的操作,如存储到数据库、解密发送到用户电子邮箱或者客户浏览器。目前的加密算法越来越多,越来越复杂,但一般数据加密的过程是可逆的,也就是说能进行加密,同时也能进行解密。

(2) 登录:一般的应用站点都会使用登录或者注册后使用的方式,因此,必须对用户名和匹配的密码进行校验,以阻止非法用户登录。在进行登录测试的时候,需要考虑输入的密码是否对大小写敏感、是否有长度和条件限制,最多可以试多少次登录,哪些页面或者文件需要登录后才能访问/下载等。

(3) 超时限制:Web 应用系统需要有是否超时的限制,当用户长时间不作任何操作的时候,需要重新登录才能使用其功能。

(4) SSL:越来越多的站点使用 SSL 安全协议进行传送。SSL(Security Socket Layer,安全套接字协议层)是由 Netscape 首先发表的网络数据安全传输协议。SSL 是利用公开密钥/私有密钥的加密技术(RSA),在位于 HTTP 层和 TCP 层之间,建立用户与服务器之间的加密通信,确保所传递的信息的安全性。SSL 是工作在公共密钥和私人密钥基础上的,任何用户都可以获得公共密钥来加密数据,但解密数据必须要通过相应的私人密钥。进入一个 SSL 站点后,可以看到浏览器出现警告信息,然后地址栏的 http 变成 https,在做 SSL 测试的时候,需要确认这些特点,以及是否有时间连接限制等一系列相关的安全保护。

(5) 服务器脚本语言:脚本语言是最常见的安全隐患,如有些脚本语言允许访问根目录,经验丰富的黑客可以通过这些缺陷来攻击和使用服务器系统,因此,脚本语言安全性在测试过程中也必须被考虑到。

(6) 日志文件:在服务器上,要验证服务器的日志是否正常工作,例如 CPU 的占用率是否很高、是否有例外的进程占用、所有的事务处理是否被记录等。

(7) 目录:Web 的目录安全是不容忽视的一个因素。如果 Web 程序或 Web 服务器处理不适当,通过简单的 URL 替换和推测,会将整个 Web 目录完全暴露给用户。这样会造成很大的风险和安全隐患。可以使用一定的解决方式,如在每个目录访问时有 index.htm,或者严格设定 Web 服务器的目录访问权限,可将这种隐患降到最低程度。

6.5.3 Web 负载测试

负载测试的作用就是在软件投入使用以前或者软件负载达到极限以前，通过执行可重复的负载测试，预先分析出软件可承受的并发用户极限值和性能瓶颈，以便优化程序。Web 的负载测试是获得 Web 站点的程序性能、可靠性以及稳定性等信息的重要手段。Web 的负载测试一般使用自动化工具来实现。

6.6 数据库服务器测试

数据库服务器是构成数据库系统的重要部分，数据库系统应该包含计算机硬件、数据库、数据库管理系统、应用程序系统以及数据库管理员。

在本节中，将主要介绍数据库服务器性能测试（大数据量测试、压力测试）以及并发控制过程的测试。

6.6.1 数据库服务器性能测试

数据库服务器性能测试主要从两个方面考虑，一个是大数据量测试，另一个是大容量的数据测试。

1. 大数据量测试

数据库的容量是表征数据库服务器性能的一个重要标准，在测试中，当大数据量在数据库中存在时，系统的性能肯定会受到影响，合理的数据库服务器管理程序以及数据库结构将会将这种影响降低到最小，例如在大数据量（成千上万、几十万条记录）处理时，通过数据库表的索引定义、数据库表空间、Log 大小将会直接影响到数据库的存储/检索性能和速度，另外，数据库服务器的 CPU 占用率也会受到影响。在数据库服务器上，不应该出现由这样的操作导致数据库系统的负荷超载甚至崩溃的现象。

在进行数据库性能测试时，需要使用一个自己编写的工具软件来动态给数据库加压，从不同的数据量和执行的语句，来测试数据库的执行速度，必要时甚至需要测试 CPU/内存占用率等。可以进行如下测试设计。

（1）所有的页面可能提交 SQL 语句跟踪，例如可以通过工具来取得执行每个页面刷新或者新链接以及其他操作（删除、排序等）的 SQL 语句。

（2）向数据库里面批量加入适当的满足条件的语句。

（3）执行操作，通过监控程序来获取执行速度，进程占用 CPU 和内容信息以及其他需要分析的信息。

（4）获取测试结果，找到数据库结构或者程序需要优化的地方。

2. 大容量数据测试

在数据库性能测试过程中，还需要考虑大容量数据测试。例如在测试中，可以使用带有视频、图片的数据，假设做这样的测试，在数据库中插入 1000 个图片文件，每个图片文

件的大小为 4MB,这样数据库在插入这些数据后的容量为 1000×4MB=4000MB≈4GB,可以通过在空表中插入 1000 条数据,选择 900 条数据,更新 900 条数据,删除 900 条数据这样一个过程来进行测试,跟踪这些操作所需要的时间、进程的 CPU 占用率以及内存信息等,最后得出结果,形成表格,从而分析数据库性能情况。

数据库容量和性能测试是至关重要的,不合理的表结构以及程序中不合理的代码将使数据库的性能降低,甚至崩溃,因此,通过性能测试可以优化数据库。

6.6.2 数据库并发控制测试

数据库的并发控制能力是指在处理多个用户在同一时间内对相同数据同时进行访问的能力。常用的关系型数据库系统都具备这种能力,例如火车售票系统、银行数据系统。

并发操作带来的数据不一致性包括 3 类:丢失修改、不可重复读和读"脏"数据。

1. 丢失修改

两个事务 T1 和 T2 读入同一数据并修改,T2 提交的结果破坏了(覆盖了)T1 提交的结果,导致 T1 的修改被丢失。

2. 不可重复读

不可重复读是指事务 T1 读取数据后,事务 T2 执行更新操作,使 T1 无法再现前一次读取结果。

3. 读"脏"数据

读"脏"数据是指事务 T1 修改某一数据,并将其写回磁盘,事务 T2 读取同一数据后,T1 由于某种原因被撤销,这时 T1 已修改过的数据恢复原值,T2 读到的数据就与数据库中的数据不一致,则 T2 读到的数据就为"脏"数据,即不正确的数据。

避免不一致性的方法和技术就是并发控制。最常用的并发控制技术是封锁技术。封锁就是事务 T 在对某个数据对象例如表、记录等操作之前,先向系统发出请求,对其加锁。加锁后事务 T 就对该数据对象有了一定的控制,在事务 T 释放它的锁之前,其他的事务不能更新此数据对象。也可以用其他技术,例如在分布式数据库系统中可以采用时间戳方法来进行并发控制。

在数据库并发控制测试过程中,需要针对程序控制的流程来设计测试。测试的重点是并发控制逻辑分析以及锁控制的逻辑分析,针对程序控制的测试过程分为以下两个部分。

(1) 并发流程分析。按照数据可处理的流程来设计测试的逻辑重点,分析并发控制的点、事务锁的使用。

(2) 并发控制测试分析。按照并发控制的实现过程以及事务锁的基本机制,设计相应的测试过程以及测试用例。

当然,在并发控制测试中,可能要涉及更为复杂的测试过程,例如多线程应用程序的并发控制处理、数据的死锁控制以及死锁分析等。

6.7 基于 J2EE 平台的测试

以 Java 技术为基础的 J2EE 构架为企业提供了一个快速构造大型、可伸缩的、分布式的框架的平台，J2EE 平台应用系统的测试技术已成为必要的技术。

6.7.1 J2EE 概述

J2EE(Java 2 Enterprise Edition)平台为企业级的应用系统提供了多点分布式应用模型。J2EE 使用多层的分布式应用模型，应用逻辑按功能划分为组件，各个应用组件根据他们所在的层而分布在不同的机器上。事实上，J2EE 的出现正是为了解决两端模式(Client/Server)的弊端，在传统模式中，客户端担当了过多的功能。在这种模式中，第一次部署比较容易，但难于升级或改进，扩展性也不理想，而且经常基于某种专有的协议(通常是某种数据库协议)，使得重用业务逻辑和界面逻辑非常困难。J2EE 的多层企业级应用模型将两层模型层面分成多层。一个多层化应用能为各种服务提供一个独立的层，典型的 J2EE 四层结构如下。

- 运行在客户端机器上的客户层组件。
- 运行在 J2EE 服务器上的 Web 层组件。
- 运行在 J2EE 服务器上的业务逻辑层组件。
- 运行在 EIS(Enterprise Information System)服务器的企业信息系统层软件。

1. J2EE 应用程序组件

J2EE 应用程序是由组件构成的。J2EE 组件是具有独立功能的软件单元，它们通过相关的类和文件组装成 J2EE 应用程序，并可以与其他组件交互。J2EE 定义了 3 种 J2EE 组件：应用客户端程序和 Applets 是客户层组件。Java Servlet 和 Java Server Pages(JSP)是 Web 层组件。Enterprise Java Beans(EJB)是业务层组件。

(1) 客户层组件：J2EE 应用程序可以是基于 Web 方式的，也可以是基于传统方式的。

(2) Web 层组件：J2EE 的 Web 组件可以是 JSP 页面或 Servlets。按照 J2EE 规范，静态的 HTML 页面和 Applets 不算是 Web 层组件。Web 层可能包含某些 JavaBeans 对象来处理用户输入，并把输入发送给运行在业务层上的 Enterprise bean 来进行处理。

(3) 业务层组件：业务层代码的逻辑用来满足银行、零售、金融等特殊商务领域的需要，由运行在业务层上的 Enterprise bean 进行处理。一个 Enterprise bean 从客户端程序接收数据，如果必要的话再进行处理，并发送到 EIS 层储存，这个过程也可以逆向进行。

(4) 企业信息系统层软件。企业信息系统层软件运行在 EIS(Enterprise Information System)服务器中。

2. J2EE 客户端

J2EE 客户端可以是 Web 客户端或者应用程序客户端。

1）Web 客户端

一个 Web 客户端包含两个部分：由各种 Web 的混合语言(HTML、XML 等)组成的 Web 页面和 Web 浏览器。Web 页面通过运行在 Web 层的组件产生，Web 浏览器将负责与 Web 服务器的页面数据进行交互，Web 客户端又称之为瘦客户端，瘦客户通常只负责做一些简单的工作。执行复杂的业务规则等操作不会由瘦客户端来执行，当使用瘦客户端的时候，繁重的工作将在 J2EE 服务端执行，基于 J2EE 的服务器技术能在安全性、速度、服务、可靠性之间做出折中。

2）Applet

Web 页面可以通过嵌入的 Applet 程序来接收来自 Web 层的数据，一个 Applet 程序是由 Java 编写的小应用程序，它在 Web 浏览器上执行。

3）应用程序客户端

一个运行在客户机上的 J2EE 应用客户端，为用户提供了一种方法，可以控制一些提供任务需求的用户接口。典型的应用是使用 Swing 类或者抽象窗口工具(AWT)提供的 API，来创建用户图形接口，当然基于命令行的程序也能如此。应用客户端程序能通过运行在业务层来访问企业级的 JavaBeans，然而，如果应用程序需要授权，J2EE 程序也可以和运行在 Web 层的 Servlet 打开 HTTP 连接通信。

4）JavaBeans 组件架构

服务层和客户层必须使用基于 JavaBeans 组件架构的组件，用以管理在应用客户端和运行在 J2EE 服务器上的组件之间进行的通信，或者是服务组件和数据库之间的通信。在 J2EE 的规范中，JavaBeans 组件没有被包括在 J2EE 组件中。JavaBeans 有一些常量以及 get/set 方法来访问这些常量，JavaBeans 组件在设计和实现中是典型的、简单的使用方法，但必须符合 JavaBeans 组件结构大纲的要求来进行命名和设计。

5）J2EE 服务通信

客户层和 J2EE 服务器的业务层使用直接或者间接的方式进行通信。客户端运行在浏览器上，通过 JSP 页面或者 Servlet 来被执行。

6.7.2 基于 J2EE 应用的单元测试技术

基于 J2EE 架构的测试非常复杂，尤其是进行系统测试时，这个问题更为突出。这里只介绍利用白盒测试方法来实现 J2EE 单元测试的过程和方法。

1. 测试原则

Java 语言是一个支持面向对象的语言，通常情况下可以将程序的一个单元看成是一个独立的类，因此进行单元测试就是进行类测试。

(1) 不需要测试 get 和 set 这样的行为。

(2) 一个方法至少需要测试一次。

(3) 各种访问、修改也会对测试产生影响。

2. 测试步骤

(1) 判断组件的功能：通过定义应用系统的整体需求，将系统划分成几个对象，并且

需要十分清楚组件的基本功能。因此,J2EE 单元测试实际上也属于设计过程的一部分。

(2) 设计组件行为:依据所处理的过程,可以通过一个正规的或者非正规的过程实现组件行为的设计,可以使用 UML 或者其他文档视图来设计组件行为,从而为组件的测试打下基础。

(3) 编写单元测试程序(或测试用例)确认组件行为:这个阶段,应该假定组件的编码已经结束而组件工作正常,需要编写单元测试程序来确定其功能是否和预定义的功能相同,测试程序需要考虑所有正常和意外的输入,以及特定的方法可能产生的溢出。

(4) 编写组件并执行测试:首先,创建类及其所对应的方法标识,然后遍历每个测试实例,为其编写相应代码使其顺利通过,然后返回测试。继续这个过程直至所有实例通过。此时,停止编码。

(5) 测试替代品:对组件行为的其他方式进行考虑,设计更周全的输入或者其他错误条件,编写测试用例来捕获这些条件,然后修改代码使得测试通过。

(6) 重整代码:如果有必要,在编码结束时,对代码进行重整和优化,重整后,返回单元测试并确认测试通过。

(7) 当组件有新的行为时,编写新的测试用例:每次在组件中发现故障。编写一个测试实例重复这个故障,然后修改组件以保证测试实例通过。同样,当发现新的需求或已有的需求改变时,编写或修改测试实例并修改代码。

(8) 代码修改,返回所有的测试:每次代码修改时,返回所有的测试以确保没有打乱代码。

3. JUnit 框架简介

JUnit 是为 Java 单元测试而提供的一种框架,使得 Java 单元测试规范而有效,并且有利于集成测试。

1) JUnit 的目标

(1) 创建一个通用的测试框架,将测试代码封装到对象中,从而使开发者同步设计并配置自己的单元测试。

(2) 让测试代码不会因为时间推移而变化,具有保值性,使测试代码标准化,从而保证编写原始测试代码之外的人也可以执行和维护测试,使多人的联合测试不会导致混乱。

(3) JUnit 框架在新创建的测试用例和旧的测试用例之间起到杠杆作用,能够为重复测试提供便利。

(4) 将测试代码从系统中分离开,两者可同步发展。

2) JUnit 的框架成员的逻辑分析

(1) 被测试的对象(类、多个类、子系统)。

(2) 对测试目标进行测试的方法与过程集合,可将其称为测试用例。

(3) 测试事务的集合,可容纳多个测试用例,将其称作测试组件。

(4) 测试结果的描述与记录。

(5) 每一个测试方法所发生的与预期不一致状况的描述,称其测试失败。

(6) JUnit Framework 中的出错异常。

3）JUnit 框架功能以及原理描述

(1）JUnit 测试用例的标准用法通常是创建一个测试类，其中包含的各个测试方法能够覆盖一个正在开发的待测试类的各个功能。

(2）测试接口与测试用例、测试组件形成了复合结构，Run(TestResult）则是复合方法。可通过 add Test(Test)来容纳测试组合形成测试包。

(3）TestCase 可在框架中视为测试单元的运行实体。用户可以通过它派生自定义的测试过程与方式(单元)，利用命令模式与复合模式使其形成可组合装配的可扩展的测试批处理。

(4）用户层可通过 Java 的匿名内部类来集成化重载 runTest()形成特性测试类，同时 JUnit 3.0 版本以后则支持利用 Java 的类属性来动态框架后台生成这些特性测试单元类。

(5）Assert 类包含了 assertEquals()、assertSame()、assertTrue()等静态工具方法，为使用户对系统所了解的类型尽可能少，JUnit 框架将 Assert 作为了 TestCase 的超类，TestCase 同时继承了 Assert 的实现与 Test 接口(class Adepter pattern)。用户可在 TestCase 中直接调用这些 assertXXX()等静态工具方法。

(6）TestResult 描述了整个测试执行过程的测试结果，JUnit 框架将其作为参数传递，当测试任务执行完毕后，TestResult 内则包含了所有 TestCase 的测试结果。

(7）TestFailure 描述了测试过程中的错误信息，TestResult 通过 Vector 容纳了测试过程中生成的 TestFailure，将其作为测试结果参数依据。

4）使用 JUnit 编写测试

使用 JUnit 编写单元测试用例主要从以下几个方面考虑。

(1）设计单个测试程序。JUnit 测试不需要人工的判断和解释，而且可以在同一时间内同时执行多个测试。在实际编写 JUnit 单个测试程序中，需要按照以下的步骤进行。

- 创建一个 TestCase 的实例。
- 创建一个构造器，它能接收字符串类型作为参数并可以传送给父类。
- 重载 runTest()方法。
- 当需要检验一个值或者变量时，调用 assertTrue()方法，它将返回一个布尔型值来标识测试成功或者失败。

如果想写一个类似于已经完成的测试实例，可以使用固件；而如果需要编写多个测试实例，则需要创建测试组件。

(2）设计固件。

- 创建一个 TestCase 的子类。
- 创建一个构造器，它能接收字符串类型作为参数并可以传送给父类。
- 在固件的每个部分各加入一个实例变量。
- 重载 setup()方法来初始化变量。
- 重载 teardown()方法来释放在 setup()过程中所使用的资源。

(3）设计 TestCase。

- 在固件类中加入测试实例的方法，确认其为 public，并不能通过映射来调用。

- 创建 TestCase 类的一个实例，通过测试实例方法把名称传递给构造器。

(4) 设计 TestSuite。TestSuite 是为了同时运行多个 TestCase 而实现的。在 JUnit 中，TestSuite()、TestCase()都是类模块，其含义和前面提到的测试组件和测试用例内涵一致，但其概念不同。

6.7.3 Servlet 的单元测试

根据 Servlet 的定义，一个 Servlet 是在请求/应答环境下实现的。一个 Servlet 的 Service()方法对请求及其响应实施动作，可以使用任何可能出现的参数组装请求对象，然后检验相应对象以确保出现预计结果。

这种方法需要对请求和相应对象进行分类，并在子类中加入测试支持，如果一个 Servlet 从响应中得到输出流，可以将其存储在测试方法中检验的响应对象里。

6.7.4 JSP 单元测试

单元测试 JSP 程序比较难，尤其是该 JSP 程序里面嵌入了大量 Java 代码的时候，因为 JSP 不是一个 Java 对象，不能使用 JUnit 框架来测试，最好的办法是使用自动化工具来根据预定义的脚本检验 JSP 页面，如使用 Rational Robot。

6.7.5 数据库访问层的单元测试

测试数据库访问代码的方式较多，从带有内嵌于代码中的 SQL 语句 JDBC 到第三方软件解决方案，存在多种编写数据库访问层的方案。在这里介绍测试数据库访问代码的常用方法。

在一个应用系统中，有一些代码执行数据库访问(即读写数据库)并进行操作，例如一个数据库驱动层软件，使用 JDBC 和 Oracle 数据库相连，并从中取出数据处理到另外的一个 MySQL 的数据库中。这样，就需要考虑 JDBC 与 Oracle 以及和 MySQL 数据库的连接引擎、数据处理等一系列复杂的过程。在数据库访问测试中，有效单元测试的第一步是判断哪些组件是访问数据库的，哪些组件是用于数据处理的。任何访问数据库的组件都可以当作实用对象或事务对象来进行单元测试，测试方法和前面的描述相类似。在 JUnit 中，提供了两种运行与测试执行之前和之后的方法，即 setUP()和 tearDown()，setUP()方法运行在每个 test()方法之前，而 tearDown()运行于其后。使用 setUP()方法可以在其中启动一个事务处理，并在 tearDown()中执行 rollback 操作，这样，就不会因为测试造成数据库系统的损坏，从而产生错误输出。

由于 J2EE 应用的复杂性，基于 J2EE 的单元测试还有很多内容，如 JavaBeans、EJB 和 RMI 对象的测试，另外，J2EE 性能测试也是非常重要的一个环节，针对不同的应用服务器，如 Tomcat、Jrun、Weblogic 以及对应的应用/部署/程序的规模大小，各自的测试重点又有不同。

6.8　基于.NET的ACT

6.8.1　ACT概述

1. ACT的用途

ACT(Microsoft Application Center Test)是VS.Net自带的一个测试工具,是专为Web服务器压力测试和分析Web应用程序(包括ASP及其所用的组件)的性能而设计的。ACT通过与服务器建立多个连接并快速发送HTTP请求来模拟成员众多的一组用户。

ACT支持多种不同的身份验证方案和SSL协议,非常适用于测试个性化的安全站点。ACT的主要用途是进行持续时间长、高负载的压力测试,但可编程的动态测试对功能测试同样非常有帮助。ACT与所有使用HTTP协议的Web服务器和Web应用程序兼容。

2. ACT的功能

ACT的Visual Studio.NET Edition版本支持的测试类型如表6-1所示。

表6-1　支持的测试类型

支持的测试类型	Visual Studio.NET Edition	支持的测试类型	Visual Studio.NET Edition
动态测试	是	重复测试	否
静态测试	否		

ACT的Visual Studio.NET Edition版本的主要功能如表6-2所示。

表6-2　功能

功　　能	Visual Studio.NET Edition
允许创建并管理用户和Cookie信息	是
与Visual Studio.NET IDE集成	是
支持SSL安全性和多种身份验证方案	是
存储测试数据供以后比较和绘图	是
创建到Web服务器的可配置同时连接数	是
支持通过录制ACT浏览器会话创建测试	是

3. ACT的系统要求

1) 软件要求

ACT要求配置以下软件。

- Windows 2000(Professional、Server、Advanced Server或Data Center Server)。ACT也支持Windows XP和Windows Server。
- Internet Explorer,如果希望通过录制浏览器会话来创建测试。只有在录制测试的计算机上才需要。

2）压力测试环境

测试环境应只包含开发或测试用 Web 服务器。

- 所有与测试无关的网络活动都应降到最低程度。避免在同一测试环境中同时运行多个测试。
- 对于压力测试，应创建足够的负载，使 Web 服务器的处理器使用率至少达到 80%。
- 应使用速度快的网络组件，并避免使用 HTTP 代理服务器。压力测试依赖于负载级别的增加，直到 Web 服务器或 Web 应用程序成为瓶颈并阻止负载级别进一步增加为止。如果系统的任何部分比 Web 服务器或 Web 应用程序速度慢，则不可能测试 Web 服务器或 Web 应用程序的最大容量。即使增加测试负载级别也不能解决这个问题。

3）ACT 客户端创建的负载量估计值

ACT 客户端创建的负载量由多种因素决定。下列方案专门用于将 ACT 客户端以外出现瓶颈的可能性降至最低。以这种方法确定的 RPS（每秒请求）值可以代表理想条件下 ACT 可以生成的最大负载。

（1）ACT 客户端。

- 软件：Windows 2000 Server。
- 处理器：600MHz Pentium Ⅲ。
- 内存：128MB。
- 测试信息有 5 个请求。在每两次请求之间不使用延迟。对于动态测试，采用单个连接请求全部 5 个页面。这些页面全都是小（15B）HTML 文档。浏览器同时连接数设为 10。

（2）Web 服务器端。

- 软件：Windows 2000 Advanced Server，IIS 5。
- 处理器：650MHz Pentium Ⅲ。
- 内存：256MB。

对于这种配置，一个运行时间为五分钟的测试的每秒平均请求数大约为 445RPS。Web 服务器 CPU 的使用率大约为 40%，而 ACT 客户端 CPU 的使用率为 100%。

4. ACT 安装

安装 ACT 时，必须使用具有本地 Windows 管理员权限的账户登录。

如果要安装 Visual Studio.NET 企业版，并在安装选项屏幕中选择 ACT，可以按照对话框中的指导完成安装。

6.8.2 ACT 创建测试

1. 使用新建测试向导创建测试

创建测试的方法有多种。一种方法是复制现有的测试，然后对副本进行编辑。另一种方法是运行新建测试向导。该向导用于创建新的、原始的测试，也可用于从其他计算机导入现成的测试。

如果启动新建测试向导，执行以下步骤。

(1) 右键单击“测试”文件夹，然后选择“新建测试”命令。

(2) 向导将启动，并开始收集有关要创建的测试类型的信息。

新建测试源有创建空测试和录制新测试两种类型，如图6-2所示。

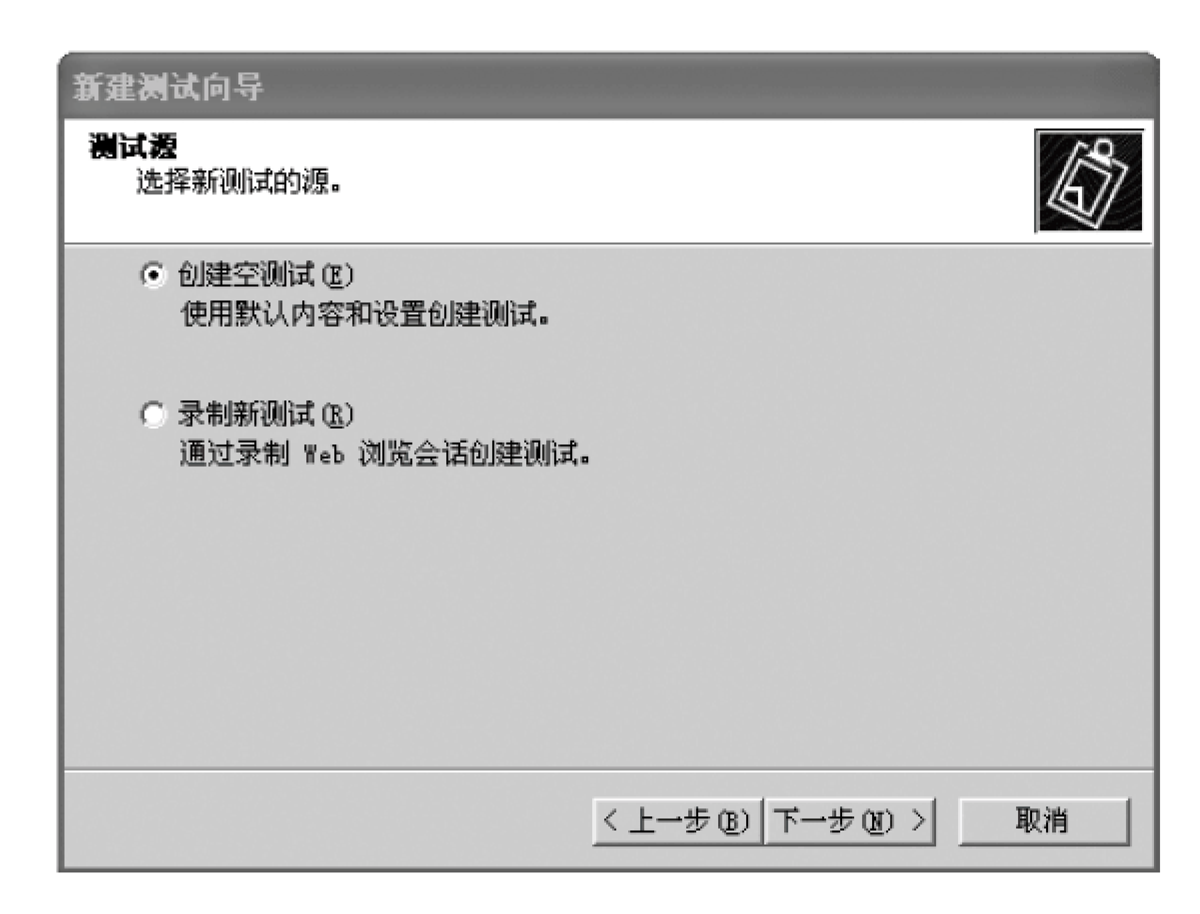


图6-2 新建测试源

如果录制了浏览器会话，在完成向导的最后一步后，就可以使用测试了。如果已创建了空动态测试，就必须使用Test对象模型对测试行为进行编程。

2. 创建空动态测试

创建空动态测试的步骤如下。

(1) 启动新建测试向导。

(2) 选中“创建空测试”单选按钮，然后单击“下一步”按钮。

(3) 从列表中选择一种脚本语言，然后单击“下一步”按钮。

(4) 在“测试名称”框中输入名称，然后单击“下一步”按钮。

(5) 单击“完成”按钮生成测试。已准备好动态测试的内容和逻辑，接下来可以进行编程。

3. 录制浏览器活动

通过录制Web浏览器会话创建测试，执行以下步骤：

(1) 启动新建测试向导。

(2) 选中“录制新测试”单选按钮，然后单击“下一步”按钮。

(3) 选择一个“测试类型”选项，然后单击“下一步”按钮。

(4) 准备开始使用Internet Explorer浏览时，单击“开始录制”按钮。

(5) 单击“停止录制”按钮可停止录制。可以多次停止和重新开始录制，避免录制浏览会话的某些部分。完成录制后，单击“下一步”按钮。

(6) 在“测试用户”对话框中，选中“自动生成用户”单选按钮，使ACT自动管理用户和Cookie。如果选中此单选按钮，测试停止后将既不保存用户账户信息，也不保存

Cookie。选中“使全局用户组与测试相关”单选按钮，可以使用自定义用户组及其相关的Cookie信息。在用户组列表中，选中要对测试启用的各组，然后单击“下一步”按钮。

(7) 在“测试选项”对话框中的“测试名称”框中输入名称，然后单击“下一步”按钮。

(8) 单击“完成”按钮后即可根据指定的设置创建测试。

(9) 应在运行测试之前编辑默认测试属性。

如果发生“拒绝访问”错误(401 HTTP 响应代码)，最常见的原因是在录制测试过程中不支持Web服务器使用的身份验证方案。解决方法是在录制过程中禁用身份验证，在运行测试之前可以重新启用身份验证。

6.8.3 ACT 测试实例

ACT可以收集性能信息，确定Web应用程序的容量，也可以创建测试，模拟同时从Web应用程序请求网页的多个用户。这些模拟测试有助于确定应用程序的稳定性、速度和响应能力。

举例说明通过录制 Web 浏览器会话来创建测试如下。

测试目的是测试字符串对象使用“＋”连接符操作和使用StringBuilder的Append方法进行字符串连接操作的区别。最终的结果相同，即生产10000个A组成的字符串。

1. 使用“＋”操作符的代码

文件名：TestStringPlus.aspx

```
private void Button1_Click(object sender, System.EventArgs e)
{
string S=string.Empty;
for(int i=0;i<10000;i++)
S+="A";

this.Label1.Text=S.ToString();
}
```

2. 使用 Append 方法的代码

文件名：TestStringAppend.aspx

```
using System.Text;

private void Button1_Click(object sender, System.EventArgs e)
{
StringBuilder S=new StringBuilder();
for(int i=0;i<10000;i++)
S.Append("A");
this.Label1.Text=S.ToString();
}
```

以上两个实例可产生同样结果，但是实现方式不同。下面用ACT来测试其性能差距。

3. 使用 ACT 进行测试

打开 Visual Studio. NET 企业版功能的 ACT,启动应用程序的步骤如下。

(1) 在"测试"项目上单击右键,选择"新建测试"命令,单击"下一步"按钮。

(2) 选择"录制新测试"单选按钮,单击"下一步"按钮。

(3) 选择自己需要的脚本语言类型,单击"下一步"按钮。

(4) 单击"开始录制"按钮,出现 IE 界面,输入网址后,录制要测试项目的操作,可以看到录制界面如图 6-3 所示,然后单击"停止录制"按钮。



图 6-3 录制测试

(5) 为这个测试项目起个名字,单击"完成"按钮,完成了一个测试项目的创建。

(6) 选择一个测试项目点,右击选择"启动测试"命令,即开始测试。

以下是对文件 TestStringPlus. aspx 的测试结果,如图 6-4 所示。

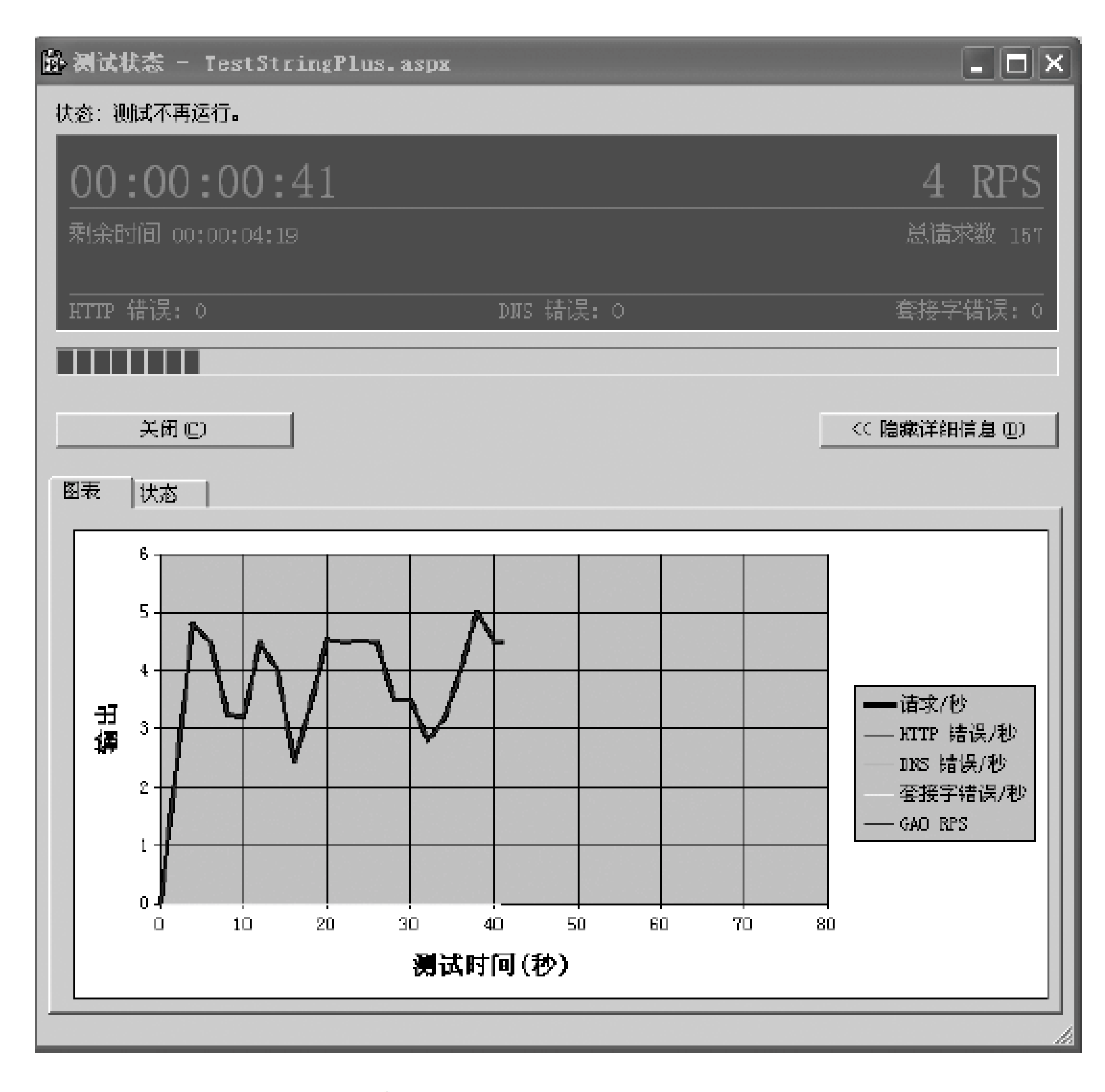


图 6-4 文件 TestStringPlus. aspx 的测试

从图 6-4 中可以看出，性能大约是 4RPS。以同样的方法测试使用 Append 方法的代码的文件 TestStringAppend.aspx 得出的结果如图 6-5 所示。

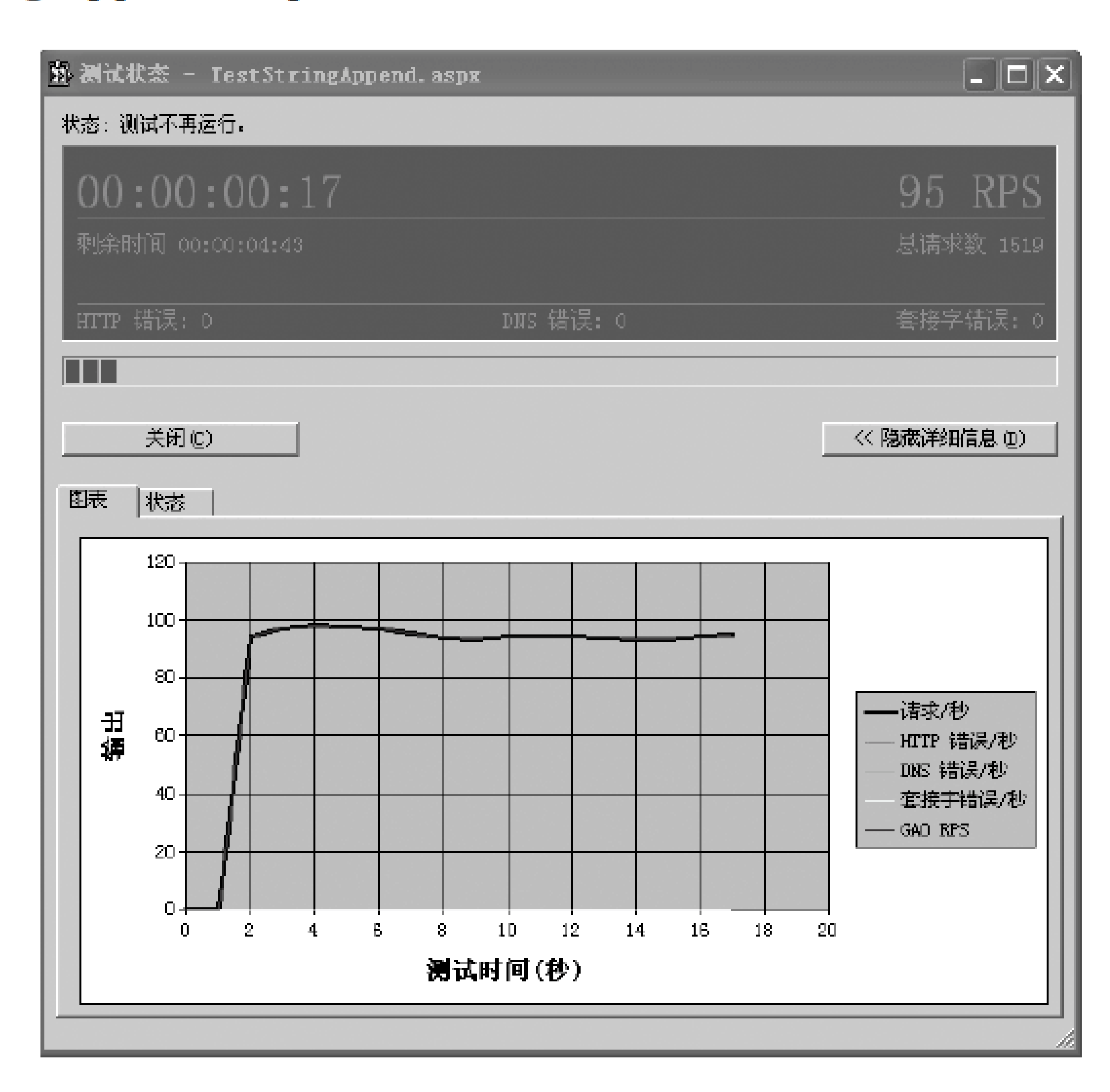


图 6-5　文件 TestStringAppend.aspx 的测试

如图 6-5 所示，性能大约是 95RPS。使用 Append 方法进行字符串连接比使用“+”操作符要高效得多(在这个例子里使用 Append 方法效率提升了 21 倍)。

小　结

Web 应用系统是一种分布并行计算系统，其应用十分广泛。在这种系统中，融合了软件开发技术、网络技术和数据库技术，对其测试，尤其对其系统性能测试复杂。通过本章内容的学习，可以理解 Web 应用系统的测试，掌握其基本测试方法，为完成实际系统测试建立基础。

习　题　6

1. 什么是应用服务器？根据应用服务器的应用范围可以将其分为哪几类？
2. Web 应用系统的三层结构分别是________、________和________。
3. 基于 Web 服务器的测试一般从哪几方面着手？
4. 简述 Web 安全性测试的重要性。
5. 利用“+”连接符操作产生 100 个 A 组成的字符串，使用 ACT 来测试其 RPS。

第7章　软件测试自动化

学习要点：

❖ 测试自动化的概念。

❖ 测试自动化的优点。

❖ 测试自动化的架构。

❖ 测试自动化的原理。

❖ 测试自动化的前处理和后处理。

软件测试自动化不仅可以减少测试开销，而且可以提高测试速度。自动测试可以在几分钟内完成需要几个小时的测试工作；测试自动化是可以重复的，可在相同的序列中使用完全相同的输入再进行测试；测试自动化可以做到即使很小的改动也可以用非常小的代价进行全面的测试；测试自动化还可以省去许多繁杂的工作。

本章介绍自动测试的概念，通过对测试执行结果的比较来说明进行测试设计更适合自动化，并介绍测试自动化的优点、存在的问题等。

7.1　测试自动化概念

测试自动化就是通过开发软件来测试软件，具体地说，测试自动化通过测试工具或其他手段，按照预定的计划对软件产品进行自动的测试，测试自动化是软件测试的一个重要组成部分，它能完成许多手工测试无法完成或者难以实现的一些测试工作。合理地、正确地应用测试自动化，能够快速、全面地测试软件，进而提高软件产品的质量、节省开支和获得高效开发。

测试自动化技术与手工测试技术不同。大量实例表明第一次构造测试自动化时比执行一次手工测试的开销大得多。但是，在测试自动化构造好之后，在运行中通常要比手工测试经济得多，其开销只是手工测试的一小部分。测试自动化的方法越好，获得的收益就越大。无论是用测试自动化，还是用手工测试执行测试的方式并不影响有效性。因此，测试自动化的程度与测试的质量独立无关。无论测试自动化做得如何出色，如果测试本身是失败的，那么测试结果也将毫无意义，测试自动化只影响测试的经济性和修改性。

用软件工具建立及维护软件测试自动化的人称为测试自动化者。测试自动化者可以不是测试者，也可以不是测试组中的成员。例如，测试组可以由具有商业知识而没有软件

开发技术的用户测试人员组成。有的时候，开发者可能需要构造及维护测试组设计的测试用例以实现测试自动化，这个开发者就是测试自动化者。

测试的质量取决于测试者实现测试质量的技术。同样，测试自动化质量也取决于测试自动化者的测试自动化技术，包括确定怎样方便地增加新的测试自动化，如何维护测试自动化以及测试自动化最终能提供什么样的效益。

软件测试自动化与测试流程、测试体系、自动化编译有关，实现测试自动化是重要技术和工具问题，需要有专门的测试团队研究和建立适合测试自动化的流程和测试体系。

7.2 测试自动化的优点

测试自动化不仅能够代替大量手工测试工作，避免重复测试，而且，它还能完成大量手工无法完成的测试工作，例如并发用户测试、大数据量测试、长时间可靠性运行测试等。与手工测试相比较具有一系列的优点，主要归纳如下所述。

1. 提高测试质量

软件开发过程就是一个不断改进的过程，每次修改都可能产生副作用，即新的缺陷。因此，当软件产品被修改，或使用环境出现变化时，对软件产品都要进行重新测试，即回归测试。通过测试来验证经修改过的软件产品质量是否符合规格说明，回归测试适合于测试自动化，测试自动化能以便利而高效的方式验证是否有新的错误进入了软件产品，可以通过测试软件产品的每个质量特性来提高软件产品的质量。

2. 提高测试效率

测试自动化可以运行更多更频繁的测试。并可以在较少的时间内运行更多的测试。通过合理地运用测试工具之后，可以减轻测试人员的手工劳动。测试工具还把控制和管理的理念引入测试过程，保证了测试进度和执行的高效率。自动化测试有助于立即测试，可以缩短开发和测试之间的时间间隔。当软件产品一完成就可以执行测试。

3. 提高测试覆盖率

利用测试自动化工具的测试功能可以提高测试覆盖率。自动化并不是开发测试用例的程序就可以了，开发完程序只是自动化的开始。测试自动化要考虑所有的活动，例如选出产品版本、选择安装、运用测试用例、生成测试数据、分析测试结果等，上述的所有活动又称为全景图，自动化测试能够覆盖全景图。

4. 执行手工测试难以实现的测试

测试自动化可以执行一些手工测试困难或难以实现的测试。例如，压力测试、负载测试、大数据量测试和崩溃性测试等，不利用测试自动化技术就不能执行。例如，如果要检查数千户登录时系统的表现，不使用自动化测试工具是不可能执行的。

5. 更好地利用全球资源

测试自动化技术可以按计划自动地进行测试，能充分利用资源，测试自动化技术可以一天 24 小时随时进行。还可以使位于不同地方、不同时区的团队监视和控制测试，继而

提供全时区的覆盖。

6. 增进了软件开发人员和测试人员的合作

为了应用测试自动化技术，需要测试人员对软件开发技术有较深入的了解，测试自动化为测试人员与软件开发人员协同工作提供了一种便利的手段和机会。

总而言之，测试自动化通过较少的开销可以获得更彻底的测试，进而提高产品的质量。

7.3　测试自动化的过程

测试自动化过程包括测试需求分析、制定测试计划、设计测试用例、执行测试、撰写测试报告、消除软件缺陷、评估测试结果等几项操作。测试自动化过程如图 7-1 所示。

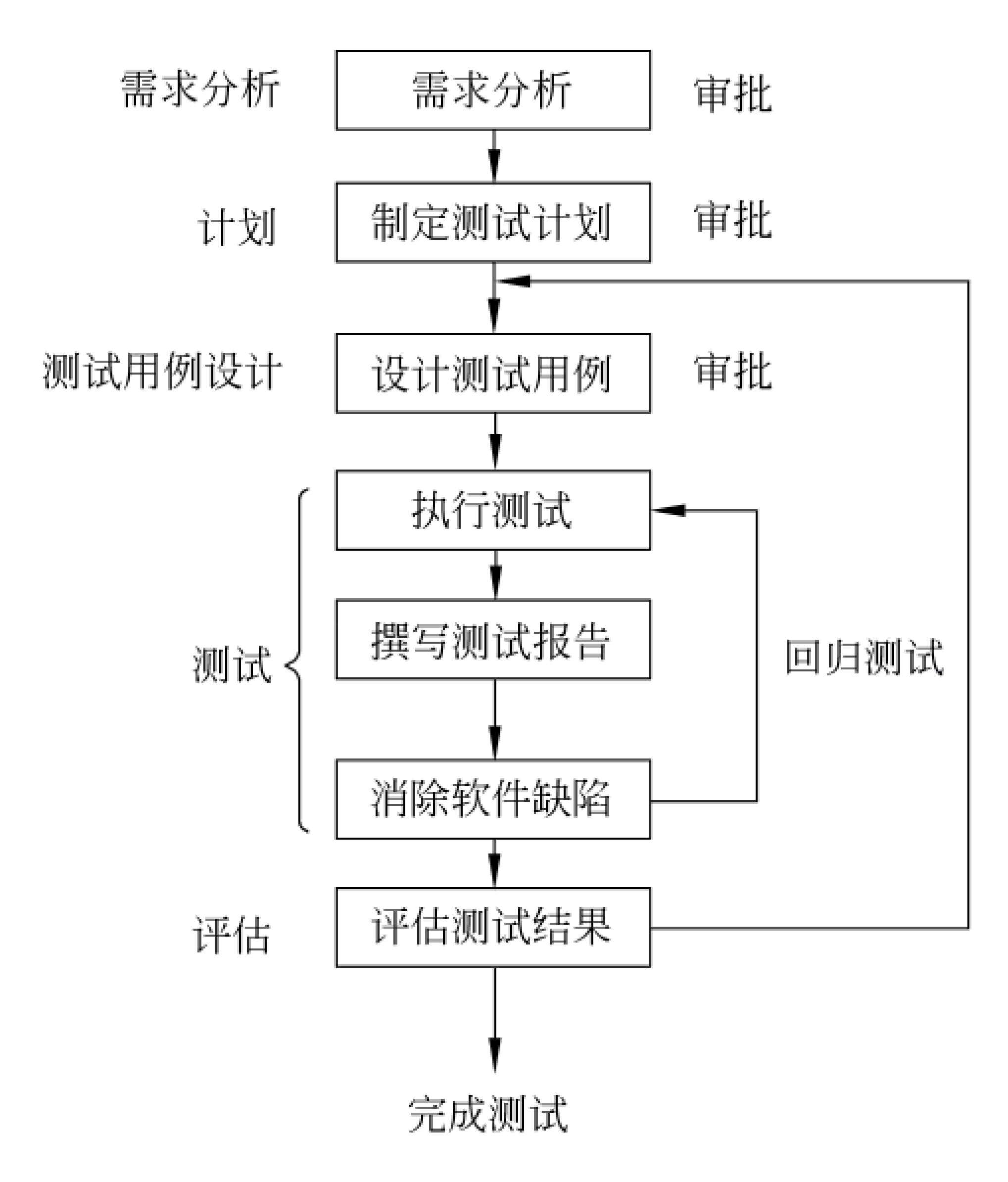


图 7-1　测试自动化过程

1. 需求分析

分析被测软件是否满足测试自动化的条件，如果满足，则进行测试自动化的需求分析，并将测试结果写入测试需求说明书中。关于测试自动化的条件可以从下述几方面考虑。

(1) 对软件项目中的相对稳定的模块可以进行测试自动化，而对于变化较大的模块可以采用手工测试。

(2) 项目的周期较长，则适合测试自动化，这时可以有充足的时间去确定测试自动化的需求、构建框架设计和编写测试脚本等。

(3) 手工测试无法完成。

(4) 资源充足、软件产品的结构相对稳定。

所有的测试活动都可以以手工方式完成。所有的测试活动也可以在一定程度上由于工具的支持而获益，但应该将最可能进行自动化处理的活动进行自动化。

2. 制定测试计划

确定测试自动化的范围和测试用例、测试数据，并形成相应的测试文档。建立测试框架，具体包括确定测试技术、定义测试体系结构、建立测试程序与测试需求之间的联系、测试数据影射、测试阶段划分、类型及测试方法等。

3. 设计测试用例

在这一步中，要确定在测试自动化的框架中，需要调用哪些文件、结构、调用过程、文件结构划分等内容。

4. 执行测试

当所有的条件准备好之后，就可以执行测试。

5. 撰写测试报告

执行测试后，填写测试用例表，并将其文档化。

6. 除掉软件缺陷

根据测试结果，修改软件，消除缺陷。如果测试发现软件有错误，软件修改后需要重新执行测试和比较活动。如果是由于环境原因导致测试失败，如使用了不正确的测试数据，则需要重新执行测试建立，执行和比较活动。如果在不同的平台运行测试，那么，要在每个平台重复这 3 个相同的操作。当软件修改时，为确保不引起其他错误，需要进行回归测试。回归测试将重复执行和比较（也可能包括测试建立）。总之，重复的活动特别适合进行自动化处理。

7. 评估测试结果

评估测试结果的流程如图 7-2 所示。

测试用例
测试报告
详细报告
测试评估
评估结果
输出：
测试结果日志
测试度量
缺陷跟踪报告

图 7-2 评估流程

7.4 测试自动化的问题

测试自动化能通过较少的开销获得更彻底的测试，同时能够提高产品的质量。但是，在实际使用测试自动化的过程中，还存在下述的问题。

（1）无论测试工具多么好，都不可以解决目前遇到的全部问题。

（2）在缺乏测试实践的情况下的紧要任务是改进测试的有效性，而不是采用自动测试。

（3）测试自动化执行工具是回归测试工具，是重复已经运行过的测试，但并不是用来发现新的软件缺陷。

（4）因为测试自动化不可能全面或测试本身就有缺陷，测试软件没有发现任何缺陷并不能说明软件没有缺陷。

（5）软件经过修改之后，一般需要修改部分或全部测试，以便可以重新正确地运行，测试自动化也需要如此。测试维护的开销将影响自动测试的积极性。

（6）商用测试工具是由销售商销售软件产品，销售商往往不具备解决问题的能力和有力的技术支持，因此用户认为测试工具不能很好地测试。

（7）测试自动化实施需要管理支持及组织艺术，必须进行选型、培训和实践、普遍使用工具。

7.5　测试自动化的局限性

测试自动化不可能完全取代手工测试。测试自动化的局限性体现在以下几个方面。

1. 不能取代手工测试

一些测试采取手工测试比测试自动化要快捷简单，因为进行测试自动化的开销较大。在以下情况下不适合自动化：测试很少运行；软件不稳定；结果很容易通过人工验证的测试，但自动测试可能很困难甚至是不可能的；设计物理交互的测试。

完全进行测试自动化没有必要，只有当测试需要频繁运行时，才需要进行测试自动化。好的测试策略应该还包括探索性或横向测试，此类测试最好由手工完成或至少先进行手工测试。当软件不稳定时，手工测试可以很快地发现这些缺陷。

2. 手工测试比测试自动化发现的缺陷更多

一般先进行手工测试，后进行测试自动化。一旦建立测试自动化用例并可以运行，那么这些测试以前在验证正确性时肯定已经运行过，因此软件在此次运行中暴露的缺陷要少得多。测试执行工具不是测试工具，而是再测试工具，即回归测试工具。

3. 对测试结果的依赖极大

工具只能判断实际结果与期望结果之间的区别（即两者的比较）。因此在自动测试中，测试的任务就变为验证期望输出的正确性。通常测试工具报告所有测试都通过，实际上只是实际结果与期望结果匹配。

4. 测试自动化并不能提高有效性

自动测试并不比手工测试更加有效。自动测试可以提高测试效率，但不可能提高测试的有效性。

5. 测试自动化受制于软件开发

测试自动化比手工测试更脆弱。被测软件部分改变有可能使测试自动化软件崩溃。尽管可以通过一些技术帮助用户建立更为健壮的测试自动化，但与手工测试比较，测试自动化受到软件的变化的影响会更大。

6. 工具本身不具有想象力

工具是按照指令执行的软件。执行一组测试可以用工具实现，也可以手工实现，但可以使用不同的方式完成相同的任务。手工测试可使测试者用创造力和想象力改进测试，可能背离原计划，也可能给测试增加一些附加内容，具有相当大的灵活性。

手工测试比测试工具优越的另一个方面是可以处理任意事件。例如，数据库连接中断，这时候必须重新建立连接，手工测试在测试期间就可以尽可能地应变情况解决异常问题。而此类的事件却可能终止测试自动化的执行。

7.6 测试自动化设计

7.6.1 测试自动化的基本架构

测试自动化的基本结构如图 7-3 所示。

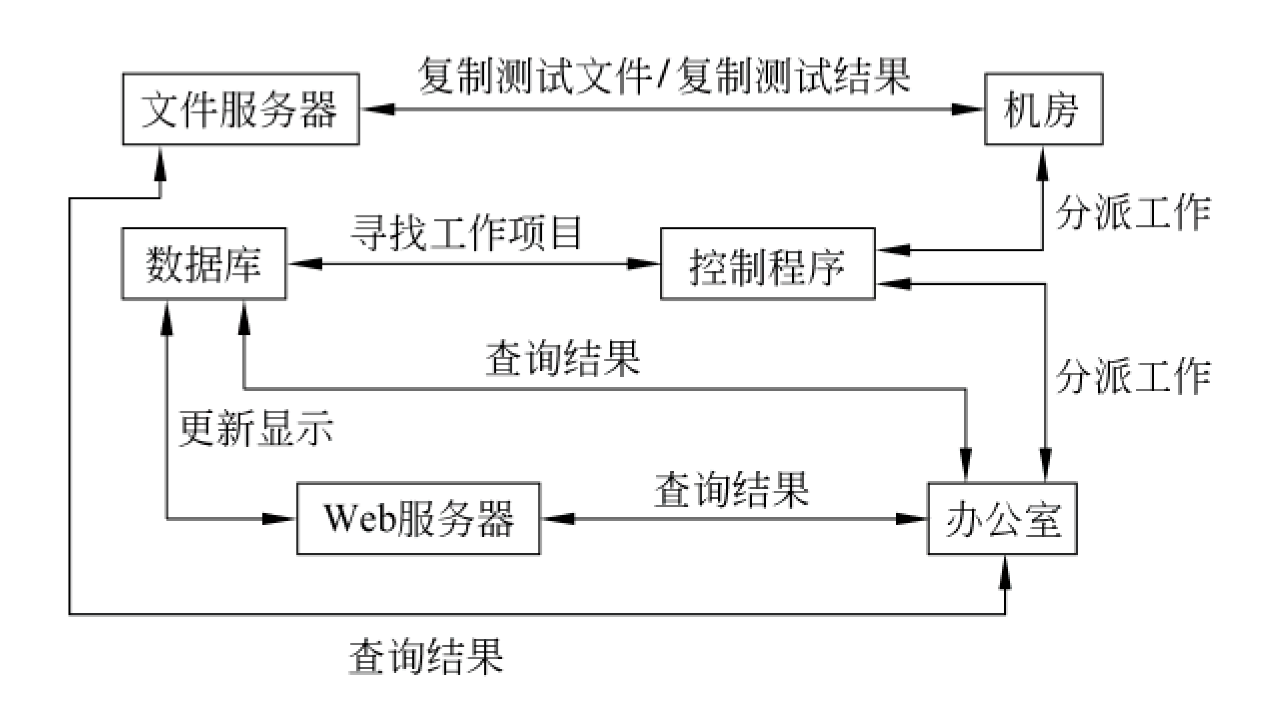


图 7-3 测试自动化的基本结构

1. 构建文件服务器

构建文件服务器，在这个文件服务器上，存放程序软件包和测试软件包。并使测试工具可以存取这些软件包。

2. 存储测试用例和测试结果

将测试用例、测试结果和生成统计所需要的数据存储在数据库服务器中。同时生成所需的统计数据。

3. 执行测试的运行环境

执行测试的运行环境是指一组用于测试的服务器或一台 PC，其中单元测试和集成测试可用单机完成，但系统测试和回归测试需要在多台计算机组成的网络环境下运行。

4. 控制服务器

控制服务器的主要功能是负责测试的执行、调度、从服务器中读取测试用例、向测试环境中的代理发布命令等。

5. Web 服务器

Web 服务器负责显示测试结果、生成统计报表、结果曲线等，接收测试人员的指令，向控制服务器传送。根据测试结果，自动发出电子邮件给测试和开发的相关人员。Web 服务器能使开发团体的任何人员都可方便地查询测试结果。

6. 客户端程序

测试人员要书写一些特殊的软件来执行测试结果与标准输出的对比或分析工作，因为有部分输出内容是不能直接对比的，需要使用程序处理。测试人员在自己的计算机上安装了程序，作为客户端程序来完成这一工作。

如图7-3所示，测试工具可以放在路径任意位置运行。可以到任何路径位置取得测试用例。同时，也可将测试结果输出到任何路径上。

7.6.2 测试自动化方法

软件测试自动化的基础是可以通过设计的特殊程序模拟测试人员对计算机的操作过程、操作行为，或者对计算机程序的语法检查。软件测试自动化实现的方法主要有：直接对代码进行静态和动态分析、测试过程的捕获和回放、测试脚本技术、虚拟用户技术和测试管理技术。

1. 代码分析

代码分析是一种自动化白盒测试方法，在代码分析时主要完成下述工作。

(1) 对代码进行语法扫描，找出不符合编码规范的地方。

(2) 根据选择的质量模型评价代码的质量，生成系统的调用关系图等。

(3) 在代码生成的可执行文件中插入检测代码，随时可以知道在关键点或关键时刻某变量的值、内存和堆栈状态等。

2. 捕获和回放

捕获和回放是一种自动化黑盒测试方法。

1) 捕获

首先将用户每一步操作都记下来。记录的方式主要有：程序用户界面的像素坐标或程序显示对象(窗口、按钮、滚动条等)的位置及相应的操作、状态变化，或属性变化。所有的记录转换为一种脚本语言所描述的过程，进而模拟用户的操作。

2) 回放

将脚本语言所描述的过程转换为屏幕上的操作，然后将被测系统的输出记录下来与预先给定的标准结果进行比较，这种方法可以大大地减轻黑盒测试的工作量，并在迭代开发过程中，能够很好地进行回归测试。

3. 脚本技术

1) 脚本语言与脚本语言程序

脚本语言是一种测试工具执行的指令集合，脚本程序也是计算机程序的一种形式，可以直接编写脚本语言程序。也可以通过录制测试的操作产生，然后再修改。脚本语言程序除了含有数据和指令之外，还包含以下信息。

(1) 同步信息。

(2) 比较信息。

(3) 数据捕获与存储。

（4）从何处读取数据。

（5）控制信息。

2）主要的脚本技术

（1）线性脚本。线性脚本是录制手工执行的测试用例得到的脚本，包含所有的击键、移动和输入数据等。所有的测试用例都可以得到回放。对于线性脚本也可以加入一些简单指令，如时间等待和比较指令等。线性脚本程序适用于较简单的测试，多数用于Web页面测试、脚本的初始化和演示等内容。

（2）结构化脚本。结构化脚本程序类似于结构化程序，由分支、循环和顺序3种基本程序结构组成。结构化脚本程序具有重用性、灵活性和维护性强的特点。

（3）共享脚本。共享脚本是指某个脚本可以被多个测试用例使用，即脚本语言允许一个脚本调用另一个脚本，可以将线性脚本转换为共享脚本。

（4）数据驱动脚本。数据驱动脚本可以将测试输入存储在数据文件中，而不是存储在脚本中，这样的脚本可以针对不同的数据输入实现多个测试用例。

（5）关键字驱动脚本。关键字驱动脚本是根据驱动脚本的逻辑扩张。

4. 数据驱动

数据驱动是指从数据文件读取输入数据，通过变量的参数化将测试数据传入测试脚本，不同的数据文件对应不同的测试用例。在这种方式中，数据和脚本分离，致使脚本的利用率和脚本的维护性显著提高。

5. 关键字驱动

关键字驱动测试将逻辑按关键字分解，进而形成数据文件，关键字对应的是被封装的业务逻辑。常用的关键字有操作对象、操作和值。关键字驱动能将脚本和数据分离，界面元素和测试内部对象名分离，测试描述和具体实现细节分离，是数据驱动的一种改进。

6. 业务驱动

业务驱动可以分为输入层业务驱动、业务驱动、数据层业务驱动、性能驱动、业务层业务驱动，其过程图如图7-4所示。

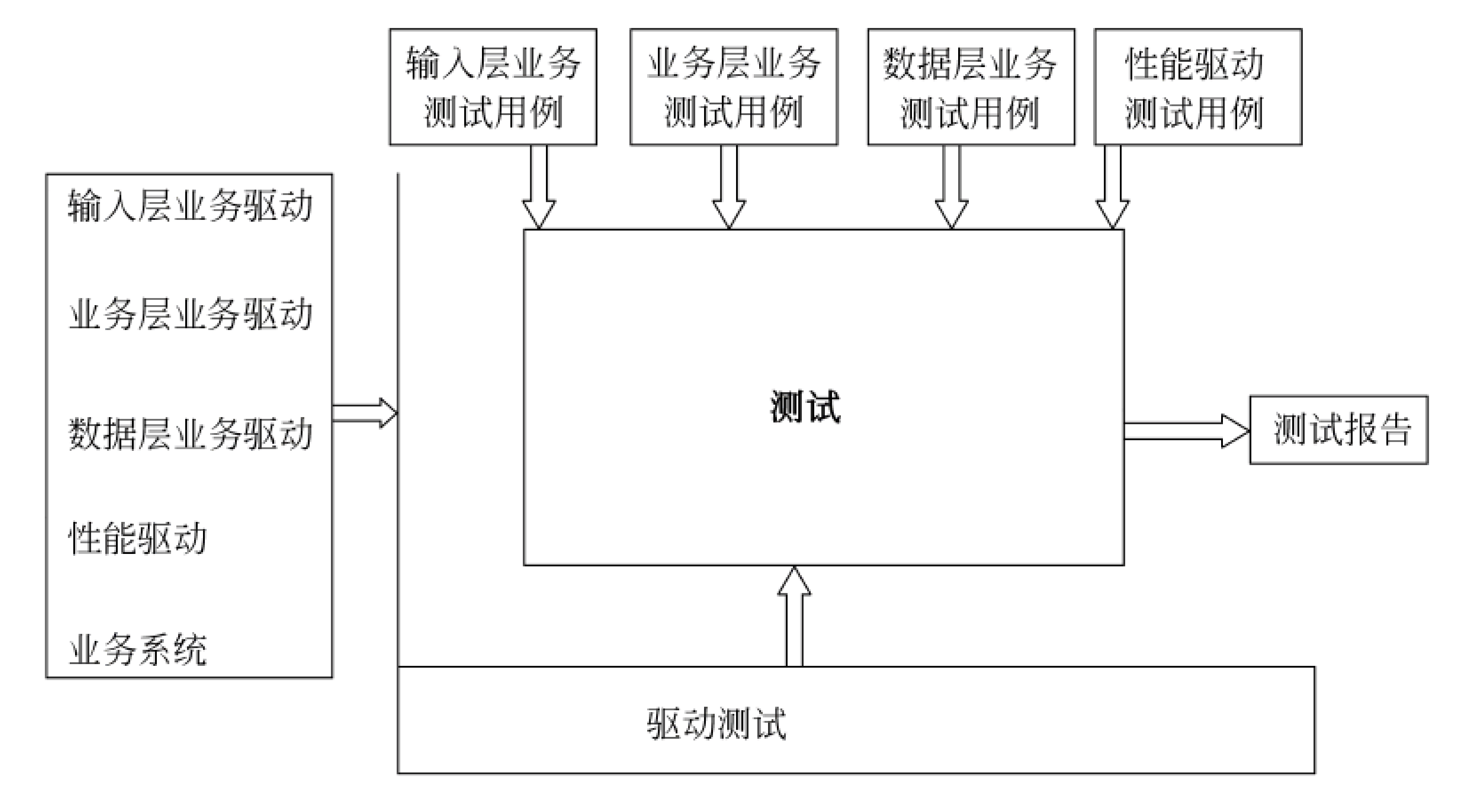


图7-4 业务驱动过程图

7. 自动比较

在测试自动化时，事先定义了期望的输出，有时将期望的输出插入到脚本程序中，然后运行脚本程序，将捕获的运行结构与期望的输出进行比较，进而决定测试用例是否通过。自动比较的内容可以是屏幕图像、窗口或窗口控件的数据或属性、网页、文件等。常用的自动比较的概念如下所述。

1）静态与动态的比较

静态比较是指在测试中并不作比较，而是将测试结果存入数据库或文件中，然后通过工具进行结果比较。而动态比较是指在测试过程中进行的比较。

2）简单比较与复杂比较

简单比较是指实际结果与期望结果完全相同可以认为一致，而复杂比较是指实际结果与期望结果有一定的差异就可认为一致。复杂比较适用于可以忽略特定差异的比较，是一种智能比较技术。

3）敏感性比较与健壮性比较

敏感性比较要求比较尽可能多的信息，例如比较整个屏幕的信息。健壮性比较是指比较少量的信息，例如只比较屏幕的最后输出结果。

4）过滤比较

过滤比较是指对期望结果和实际结果通过过滤器先进行预处理，之后再进行比较，这种方式可使测试结构准确。

8. 测试管理

测试管理的内容包括测试输入、执行过程和测试结果的管理。其中主要包含测试自动化和手工测试共同的内容，也包括测试自动化特有的内容进行跟踪、控制和管理等。特有的内容有：测试数据库、测试脚本程序、预期输出结果、测试日志等。公有的内容有：测试计划、测试用例、缺陷、测试套件等。

7.6.3　测试自动化层次

测试自动化分为以下 4 个层次。

1. 捕获和回放

捕获和回放层次是最低的层次，在这一层次中，利用测试自动化工具，能够自动生成脚本，不需要编程知识。这种方法适用于被测试的系统不变化时，小规模的自动化测试。这是由于当需求发生变化时，相应的脚本也必须重新录制。

2. 捕获、编辑和回放

使用测试自动化工具来捕获要测试的功能。通过捕获、编辑和回放，可将脚本中的任意固定测试数据（例如名字和账户等）转换为变量。这种方法能使测试脚本变得更为完善和灵活，可以大量减少脚本的数量和维护工作。但需要测试者具有一定的编程能力。

3. 编程和回放

在编程和回放层次，开发了大规模的测试套件，执行和维护测试自动化。进而重用已

有的测试用例实现测试自动化,但可构建不同的回归测试。这种方法的特点是能够在项目的早期就开始进行脚本的设计,要求测试人员具有较好的软件设计和开发能力,测试人员要进行适当的编码。

4. 数据驱动的测试

数据驱动属于专业测试,需要测试工具提供所有的测试功能。

7.7 测试自动化用例

对于任何软件系统都可有多个测试用例,但实际上只能运行其中很少的一部分测试用例,同时希望使用有限的测试用例来发现软件中的大部分缺陷,因此测试用例的选择十分重要。实践表明,随机选择测试用例不是好的测试方法,好的测试方法应该能设计和选择出好的测试用例。

7.7.1 测试自动化用例特征

测试自动化用例具有以下特征。

(1) 检测软件缺陷有效性。

(2) 测试用例的测试内容多项性。

(3) 测试用例的执行、分析和调试的经济性。

(4) 每次软件修改之后需增加测试用例的维护成本。

通常要对上述 4 个方面进行折中。例如,一些测试用例可以测试很多内容,但其执行、分析和调试的开销可能很大;也有可能在每次软件修改后需要对测试用例进行大量的维护。可见高效性有可能导致经济性和修改性较低。

因此测试技术不仅要保证测试用例具有发现缺陷的有效性,还需要测试用例经济有效。

7.7.2 测试自动化用例设计

测试用例设计可以自动生成,有许多测试工具可以进行部分测试用例自动生成。这类工具有时也称测试输入生成工具。

所有测试用例设计方法都存在一个问题,即工具可能产生大量的测试用例。工具不能区分哪些测试是最重要的,这类活动只能由测试人员完成。所有测试生成工具依赖于生成测试的算法。工具比使用相同算法的测试人员的测试更为精确,这是工具的优势。但手工测试时,可以考虑附加测试,可以对遗漏的需求进行说明,或根据个人知识指出不正确的定义。使用测试用例生成工具应该对工具可以做什么和不可以做什么有个比较清晰的认识。

下面介绍 3 种测试输入生成工具,包括基于代码的测试输入生成工具、基于界面的测试输入生成工具以及基于规格说明的测试输入生成工具。

1. 基于代码的测试输入生成工具

基于代码测试输入生成工具，如图 7-5 所示，通过检测软件代码结构生成测试输入。通过代码的路径由判断点分支确定的组成，自动生成每个路径逻辑条件的轮廓文件。这种方式与覆盖度量工具一起使用较好。

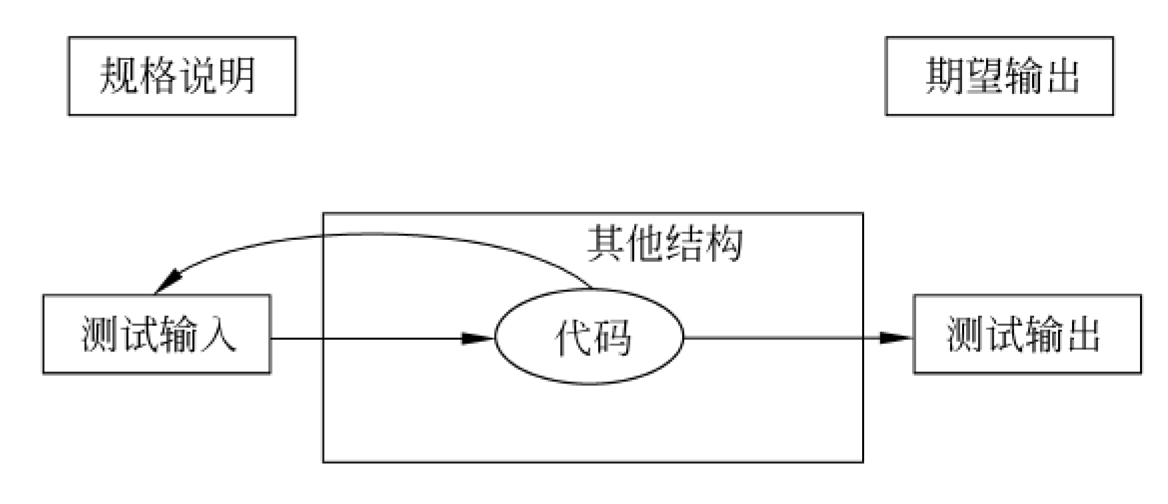


图 7-5　基于代码的测试输入生成

这种方法产生测试输入，但还需要对测试输出进行比较。基于代码测试用例设计判断软件产生的输出是否正确，只是说明代码应该做什么。这种测试方法是不全面的，因为它不能产生期望输出。

2. 基于界面的测试生成工具

基于界面的测试生成工具如图 7-6 所示，可以用于某些定义好的界面（如 GUI 或 Web）生成测试用例。如果屏幕含有各种菜单、按钮及选择框，则工具可以生成访问每个控件的测试用例。例如，工具可以测试选择框被选中时有个交叉符号，未被选中时为空（对于其他图形控件可以自动进行类型的基本测试）。再例如，选择每个域检查 Help 工作是否正常，编辑所有只显示的域，对 Help 文本进行拼写检查以及检查每个菜单项弹出的内容。工具中类似的功能还可以测试 Intranet 页面。工具可激活 WWW 页面的每个链接，然后对每页循环地做相同的测试。

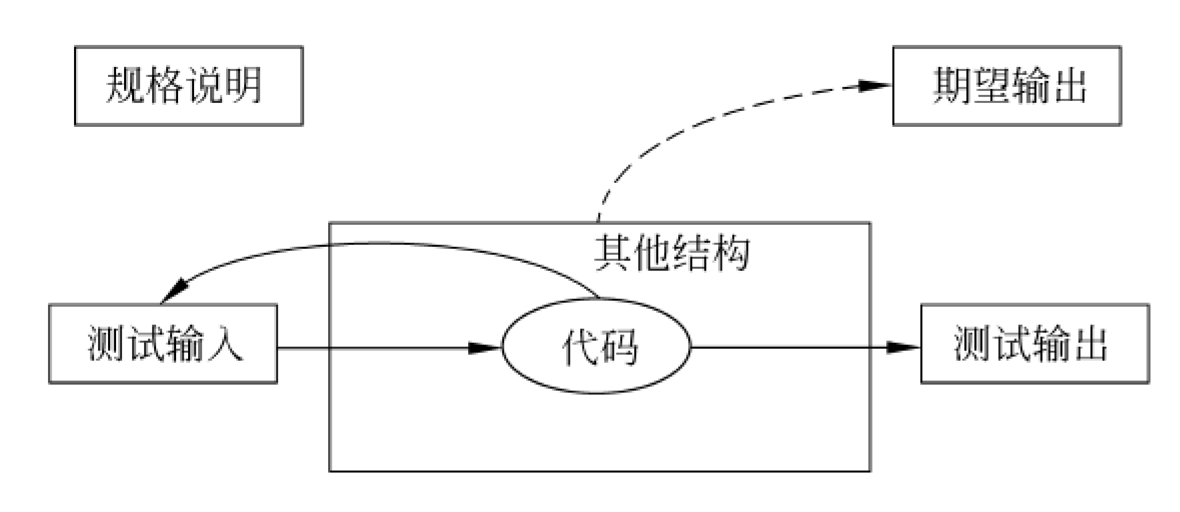


图 7-6　基于界面的测试输入和测试用例生成

基于界面的方法可以有效地发现某些类缺陷（例如，可以发现任何工作不正常的 Web 页面连接），并可以部分生成期望输出，而且是一般意义上的期望输出，即连接应该存在（正确结果）不应该断开（不正确结果）。这种自动生成工具不能判定连接是否在正确的位置。

因此，这种方法可以执行部分测试用例设计活动，产生测试输入（对工具标识的每个界面元素进行测试）以及部分期望输出，这样发现一些错误。这种方法对于执行滚动调用

测试很有效，即测试应该在某处的东西确实在某处。这种类型的测试手工进行就十分枯燥，但彻底测试却是十分必要的。

测试工具可将测试人员从繁琐的重复劳动中解脱出来，有更多的时间从事创造性劳动。

3. 基于规格说明的测试生成工具

在规格说明形式化并可被工具分析的前提下，基于规格说明的测试工具可以生成测试输入及期望输出，如图7-7所示。规格说明可以包括结构化的自然语言，如商业规则，也可以包含技术数据，如陈述和事物。如果面向对象规格说明足够严格的话，这种工具还可以进行面向对象规格说明的测试。因此这类测试工具确实具有产生期望输出的机制。

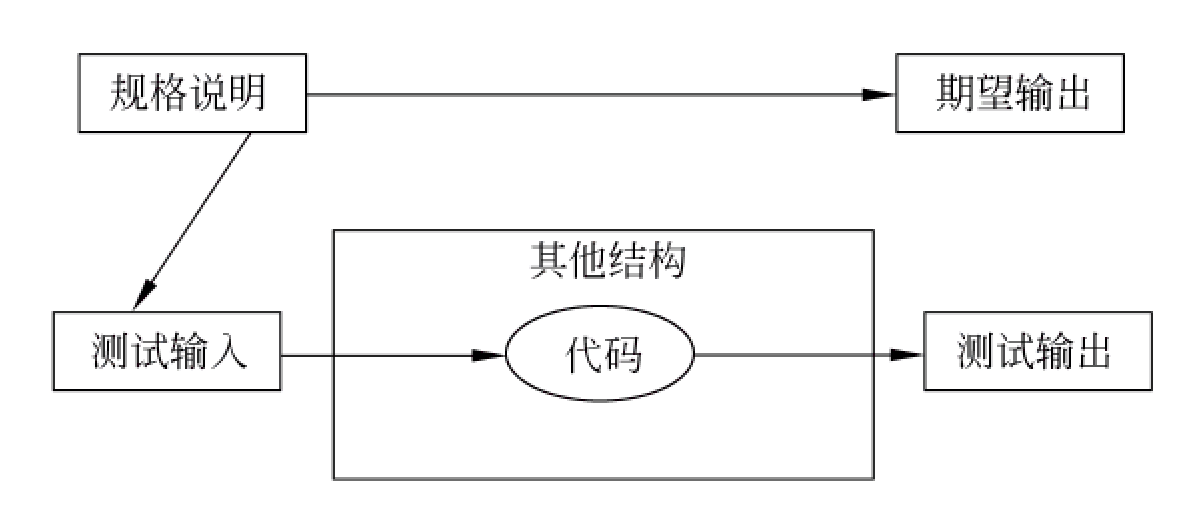


图 7-7 基于规格说明测试用例生成

例如，如果严格定义好一个输入域的允许范围，那么工具可以产生边界值以及有效等价类和无效等价类的样值。

基于规格说明的工具可以进行结构化的英文规格说明或因果图的测试，可以发现一些规格说明的缺陷，如规格说明含混或冗长。

基于规格说明方法的好处是测试检查软件应该做什么，而不是它做了什么，尽管使用现有的工具测试规格说明范围有限。从规格说明中推导测试用例越枯燥，使用这类工具的潜力就越大。

如果期望输出存储在规格说明中，并假设存储的期望输出是正确的，则可以生成期望输出。

7.7.3 测试自动化用例生成优缺点

1) 测试自动化用例生成的优点

(1) 测试自动化用例生成用于设计的繁琐部分，如激活每个菜单项或从已知的数据范围计算边界值。

(2) 可以生成针对源程序的一套完整的测试用例(代码、界面或规格说明)。

(3) 可以发现某种类型的缺陷，如丢失连接，非工作窗口项或不符合规格说明的软件。

2) 测试自动化用例生成的限制

(1) 基于代码方法不能生成期望输出。

(2) 基于界面方法只能产生部分期望输出。

(3) 基于代码和基于界面方法不能发展规格说明的缺陷。

(4) 基于规格说明的方法依赖于规格说明的质量。

(5) 所有的方法可以产生大量的测试,而实际操作起来很困难。

(6) 测试前仍然需要专家判断产生测试的必要性,并考虑任何工具都无法产生的测试。

将测试的智力部分自动化会很困难,而且测试的机械部分也不可能做到完全自动化。

7.8 测试自动化的前处理和后处理

1. 前处理

前处理即指测试工作开始之前所必须进行的处理。也就是指所有与建立和恢复这些测试先决条件相关的工作。

在大多数测试用例中,使用之前要具备先决条件。这些先决条件应被定义为每一个测试用例的一部分并在测试之前实现。例如,该测试可能需要一个数据库,其中包含某些特殊的客户记录或是一个详细的目录,存放着含有特殊信息的相关文件。

在某些测试用例中,先决条件一旦建立便可一劳永逸,因为它们在测试过程中不会改变。而在另一些测试用例中,每次测试执行过后需要进行恢复工作,因为先决条件在测试过程中发生了变化。

2. 后处理

后处理就是指在测试自动化完成后进行的工作。

测试用例一旦执行就会立即产生测试结果,其中包括测试的直接产物(当前结果)和副产物(如工具日志文件),它们所涵盖的范围可能很广,需要对这些人为产物进行处理,或者是评估测试的成败或者是进行内务处理。

有些测试结果可以清除(例如没有发现差别的差异报告),而有一些则必须保留(例如同预期输出结果不符的输出文件)。要保存的结果应该存放到一个公共的位置以便对其进行分析或只是为了防止它们被以后的测试改变或损坏。

3. 前处理和后处理任务的特征

1) 数量多

有大量潜在的前处理和后处理任务要执行。其中一部分(与测试用例相关的那部分)需要在每次运行测试用例时都执行。这通常被列在重要工作之中。自动化前处理和后处理任务可节省大量工作。

2) 成批量出现

通常会有许多待处理的前处理和后处理任务在同一时刻出现。例如,可能需要复制好几个而不是一个文件,或是要编译数个脚本。这些任务可以按种类自动化。

3) 类型重复多

在某个特定系统上进行的多项测试只需要简单的物理设置,因此可能只存在少数几种不同类型的前处理和后处理行为。不同测试用例之间的许多变化源自所使用的数据不

同。例如,许多对一个依赖数据库信息的系统的测试用例需要一个已有的数据库,并从数据库中提取数据,但是每个测试用例又需要不同的数据。一旦一个任务被自动化,那么许多任务也同样可以被自动化。

4) 容易自动化

这些任务通常是简单的函数,可以用一个简单的指令或命令来实现。许多复杂的函数可以减缩成用一个命令文件就可以执行的简单命令。可以用一个简单的机制来自动化所有的前处理和后处理。

4. 前处理和后处理任务的内容

1) 前处理

(1) 创建。用于为测试建立适当的前提条件,例如建立一个数据库并向其中填充测试所需的数据。有些前提条件需要某些必要数据,而另一些条件要求某些数据不存在。这种情况下,前处理任务可能包括从数据库中删除不必要的记录或从目录中清除文件。

(2) 检验。自动化所有的设置任务是不太可能的(如释放足够的磁盘空间),但检验特定的前提条件是否满足是可行的。例如,检验必需的文件是否存在,而不应该存在的文件是否真的不存在。其他还包括环境检验,像操作系统版本号,检验局域网是否运转正常,还有检验光驱中是否有可写的盘。

(3) 改造。这同创建任务类似,但特指的是那些从一处到另一处复制或移动文件的任务。例如,当一个测试需要改变一个数据文件时,需要将该数据文件从它的存储目录中复制到工作区域中,这将确保该数据文件的主副本不被测试所破坏。

(4) 转换。将测试数据保存为测试所需要的格式并不总是很方便和必要的。例如,较大的文件最好以压缩格式存储,而为了维护方便,非文本格式文件(如数据库和电子表格文件)最好以文本格式存储。

2) 后处理

(1) 删除。测试执行后的清理工作,如删除数据库中的记录。一些测试用例会产生大量的输出,可能其中只有一小部分是用来进行比较的,其他是无用的。

(2) 检验。测试用例所期望的结果。如确认一个测试用例的运行结果是期望数据库中存在的某一项。这些检验任务可以被自动化并归入后处理。

(3) 重组。与上述的删除任务类似,复制和移动文件的任务是一项简单的任务,可以被自动化处理。

(4) 转换。将不利于进行比较和分析的结果进行格式转换。例如将大量的数据库的记录项整理成图标形式分析。

小　　结

自动化测试与测试不是一个概念。尽管没有一种工具可以使任何测试活动完全自动化,但工具可以对整个开发周期中的所有类型的活动进行支持。

测试自动化具有一些显著的特点得到广泛认可,如果多次执行一致的回归测试、完成

手工测试难以实现的测试，可以更好地利用资源，提高测试重用性，提前进入市场及提高可信度。但测试自动化也存在着一些问题，如不现实的期望，较差的测试经验，无安全性意识，维护成本以及其他技术管理方面的问题。

测试过程中的活动并不都适合进行测试化。尽管有几种支持测试用例设计活动的方法，测试中重复而烦琐的活动比智力活动更适合进行自动化。

测试自动化具有一定的局限性，并不能取代手工测试。手工测试可以比测试自动化发现更多的缺陷。测试自动化对期望的结果的正确性依赖极大，测试自动化并不能改进测试有效性，并对软件开发有一定的制约作用。测试工具缺乏创造性且灵活性较差。但是必须认识到，恰当地进行测试自动化可以大大促进软件测试的质量和产品化。

习　题　7

1. 生存周期测试的工具有哪些？各自的功能是什么？
2. 列举在测试自动化构造好后，相比手工测试的优点。
3. 列举测试自动化比手工测试不足的地方。
4. 对于测试自动化的说法，以下正确的是(　　)。
   A. 测试自动化比手工测试更能发现软件中的潜在缺陷
   B. 运行了测试自动化，就不需要再进行手工测试了，因为测试自动化一定可以发现手工测试发现的错误
   C. 软件修改了已经发现的错误之后，需要修改测试
   D. 商用的测试执行工具可以测试出所有的软件问题
5. 自动生成测试用例的工具有基于________的输入生成工具，基于________的输入生成工具和基于________输入生成工具。

第 8 章　软件质量与质量保证

学习要点：

❖ 软件质量的定义、因素与保证。

❖ 软件评审内容。

❖ 软件质量保证的标准。

❖ 软件质量框架。

不论什么产品，质量都是极端重要的。软件产品是逻辑产品，其特点是研发周期长，耗资巨大，软件的生产过程就是复制过程，因此，在研发软件产品时，必须特别注重软件质量。软件质量保证和软件质量控制都是软件质量管理的一环，软件质量保证的目标是预防缺陷和错误的发生，是属于防御性的方法。而软件控制是找出缺陷和错误，是属于主动出击的方法。

8.1　软件质量的定义

ANSI/IEEE Std 729—1983 定义软件质量为“与软件产品满足规定的和隐含的需求的能力有关的特征或特性的全体”。软件质量具有以下特点。

(1) 软件需求是度量软件质量的基础，不符合需求的软件就没有质量。

(2) 在各种标准中定义了开发准则，用来指导软件人员用工程化的方法开发软件。如果不遵守这些开发准则，软件质量就得不到保证。

(3) 有一些隐含的需求往往没有明确地提出来。例如，软件应具备良好的可维护性。如果软件只满足那些精确定义了的需求而没有满足这些隐含的需求，软件质量也不能保证。

软件质量是用户满足程度的描述和各种特性的复杂组合，随着应用的不同而异，随着用户提出的质量要求不同而不同。因此，有必要介绍各种质量特性及评价质量的准则。

8.2　影响软件质量的因素

对软件开发项目提出的要求往往只强调系统必须完成的功能、应该遵循的进度计划，以及生产这个系统花费的成本，却很少注意在整个生存周期中软件系统应该具备的质量

标准。这种做法的后果是质量得不到保证，使系统的维护费用昂贵，为了把软件系统移植到另外的环境中，或者使系统和其他系统配合使用，都必须付出很高昂的代价。

1. 主要因素

虽然软件具有难于定量度量的软件属性，但是仍然能够提出许多重要的软件质量指标。

从管理角度对软件质量进行度量，可以把影响软件质量的主要因素分成以下13类。

(1) 正确性：系统满足规格说明和用户目标的程度，即在预定环境下能正确地完成预期功能的程度。

(2) 健壮性：在输入的数据无效或操作错误等意外环境下，系统能做出适当响应的程度。

(3) 效率：为了完成预定的功能，系统需要的计算资源的多少。

(4) 安全性：对未经授权的人使用软件或数据的企图，系统能够控制的程度。

(5) 可用性：系统在完成预定应该完成的功能时令人满意的程度。

(6) 风险：按预定的成本和进度把系统开发出来，并且受用户所满意的概率。

(7) 可理解性：理解和使用该系统的容易程度。

(8) 可维修性：诊断和改进正在运行现场发现的错误所需要的工作量的大小。

(9) 适应性：修改或改进正在运行的系统需要的工作量的多少。

(10) 可测试性：软件容易测试的程度。

(11) 可移植性：把程序从一种硬件配置和软件系统环境转移到另一种配置和环境时，需要的工作量的多少。一种定量度量的方法是：程序设计和调试的成本与完成移植的费用之比。

(12) 可再用性：在其他应用中该程序可以被再次使用的程度。

(13) 互运行性：把该系统和另一个系统结合起来的工作量的多少。

软件产品的质量是软件工程的开发工作的关键问题，也是软件工程生产中的核心问题。计算机软件质量是计算机软件内在属性的组合，包括计算机程序、数据、文件等多方面的可理解性、正确性、可用性、可移植性、可维护性、可修改性、可测试性、适应性、再用性、完整性、适用性、健壮性、可靠性、效率与风险等多方面特性。

在软件项目的开发过程中，往往强调软件必须完成的功能、进度计划、花费成本，而忽略了软件工程生存周期中各阶段的质量标准。对软件质量的看法与提高软件质量的途径在软件工程行业中存在着不同的看法与做法，发展的趋势是从研究管理问题、产品问题转向过程问题(开发模型、开发技术)，使单纯的测试、检验、评价、验收融入设计过程中。

2. 质量评价原则

(1) 应强调软件总体质量是低成本、高质量，而不应片面强调软件正确性，忽略其可维护性与可靠性、可用性与效率等。

(2) 应在软件工程化生产的整个周期的各个阶段都注意软件的质量，而不能只在软件最终产品验收时注意质量。

(3) 应制定软件质量标准，定量地评价软件质量，使软件产品评价执行评测结合，以

测为主的科学方法。

8.3 软件质量保证

8.3.1 软件质量保证概念

软件质量保证(Software Quality Assurance,SQA)通过建立一套有计划的系统方法,来向管理层确保拟定出的标准、步骤、实践和方法能够正确地应用于所有项目。SQA 由系统性的活动组成,为软件产品的可用性提供了保证。SQA 通过使用已建立的质量控制活动过程来保证软件的一致性,并延长软件的使用寿命。SQA 由一系列评估被开发或被生产产品过程的活动组成,确保使用正确的工具、步骤和技术开发产品,最终达到防止错误出现或尽早地检查出错误,从而开发出高质量的软件。

SQA 不同于软件测试,SQA 评估过程质量,其主要目的是预防软件的缺陷。SQA 通过评审测试结果和根据对软件质量的度量监控测试的有效性,通过对软件测试文档的审核来确定测试活动是否符合建立的标准和规范。归纳为下述四条基本目的。

(1) 保证工作有计划地进行。

(2) 验证软件产品是否遵循标准、步骤和需求。

(3) 将 SQA 的结果通知给相关小组和个人。

(4) 高级管理层接触项目内部不能解决的问题。

8.3.2 软件质量保证策略

为了在软件开发过程中保证软件的质量,主要采取下述措施。

1. 审查

审查就是在软件生存周期每个阶段结束之前,都正式使用结束标准对该阶段生产出的软件配置成分进行严格的技术审查。

审查小组通常由 4 人组成:组长、作者和两名评审员。组长负责组织和领导技术审查,作者是开发文档或程序的人,两名评审员提出技术评论。评审员应由与评审结果利害攸关的人担任。

审查过程的步骤如下。

(1) 计划:组织审查组,分发材料,安排日程等。

(2) 概貌介绍:当项目复杂庞大时,可由作者介绍概况。

(3) 准备:评审员阅读材料取得有关项目的知识。

(4) 评审会:目的是发现和记录错误。

(5) 返工:作者修正已经发现的问题。

(6) 复查:判断返工是否真正解决了问题。

至少在生存周期每个阶段结束之前,应该进行一次正式的审查,在某些阶段中可以进行多次审查。

2. 复查和管理复审

复查即是检查已有的材料，以确定某阶段的工作是否能够开始或继续。每个阶段开始时的复查，是为了肯定前一个阶段结束时的审查，已经具备了开始当前阶段工作所必需的材料。管理复审通常指向开发组织或使用部门的管理人员，提供有关项目的总体状况、成本和进度等方面的情况，以便从管理角度对开发工作进行审查。

3. 测试

测试就是用已知的输入在已知环境中动态地运行系统或系统的部件。如果测试结果和预期的结果不一致，则表明系统中可能出现了错误。测试过程中产生的基本文档如下。

（1）测试计划：通常包括单元测试和集成测试，确定测试范围、方法和需要的资源等。

（2）测试过程：详细描述和每个测试方案有关的测试步骤和数据，包括测试数据及预期的结果。

（3）测试结果：把每次测试运行的结果归入文档，如果运行出错，则应产生问题报告，并且通过调试解决所发现的问题。

8.3.3　SQA 小组的任务

SQA 小组的职责是辅助软件开发小组获得高质量的软件产品。SQA 小组负责审计产品线的质量活动并根据偏差向管理者提出警告。SQA 小组执行计划、监督、记录、分析及报告的活动，主要完成以下工作。

（1）为软件项目制定一份 SQA 计划。在整个项目计划的早期阶段，制定 SQA 计划，通过组织和个人评审该 SQA 计划，对 SQA 计划进行管理和控制。组织和个人指本项目软件经理、顾客的 SQA 代表等。

（2）主要活动。按照 SQA 计划，SQA 小组进行的主要活动如下。

① SQA 小组职责和权利。

② SQA 小组的资源要求。

③ SQA 小组进度表和资金。

④ SQA 小组参加开发计划的标准和规章的情况。

⑤ SQA 小组对执行软件的评价。

⑥ SQA 小组进行评审和审计基础的项目标准和规程。

⑦ 对不符合问题建立文档和进行跟踪直至结束的规程。

⑧ 规程可作为计划的一部分。

⑨ 要求 SQA 小组生成的文档。

⑩ SQA 小组给软件工程组和其他软件相关组提供反馈信息的方法和频率。

（3）参与软件开发过程描述。

① 对组织方针的符合性。

② 对外部强加的标准和要求的符合性。

③ 适合项目使用的标准。

④ 在软件开发计划中应阐述的专题。

⑤ 项目指定的其他领域。

(4) 评审软件工程活动。

(5) 审计指定的软件工作产品以验证符合性。

(6) 按照规程，对在软件活动和软件工作产品中所鉴别出的偏差建立文档并加以处理。

(7) 顾客的 SQA 人员一起对它的活动和发展进行定期评审。

(8) 定期向软件开发组报告其活动的结果。

8.4　软件质量保证活动

图 8-1 表明了质量保证的内容。质量保证是复审、开发方法、配置控制与程序测试的综合应用。简单地说，软件的开发方法应该符合规定的软件开发规范；计划和开发时期各个阶段的工作都要进行复审；每个阶段产生的文档都必须严格管理，以确保文档和程序的完整性与一致性；作为最后和最重要的一道防线，还要坚持对程序进行各个层次的测试。

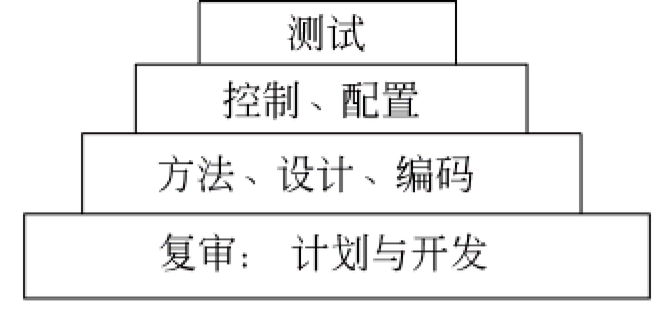


图 8-1　质量保证活动内容

所有以上的各项活动内容，都需写入质量保证计划，并由质量保证小组监督实施。由此可见，质量保证既是技术活动，也是管理活动。

1. 验证与确认

验证是为了确定开发时期中某一阶段的产品是否达到了阶段对它的需求，确认则是在整个开发结束时对所开发的软件能否满足软件需求的总评价。换句话说，前者仅要求两个相邻阶段间的一致性，后者则要求在整个开发时期内的一致性。

具体地说，验证将包含在开发时期各个阶段进行的复审、人工复查与测试活动；确认主要指测试阶段的确认测试和验收时的系统测试等活动。两者结合起来，就构成质量保证的中心内容。测试阶段使用的两种文档，测试计划和测试报告。在实际执行中，常常把上述的计划和报告扩充为关于验收与确认的计划和报告，用以代替范围较小的测试文档。

2. 开发时期的配置管理

虽然维护时期坚持配置管理十分重要。但事实上，对配置的控制从计划时期就开始了，一直延续到生存周期结束、软件停止使用后才终止。

软件配置包括生存期中各个阶段产生的文档和程序。这些文档或程序是随着软件的开发进程逐步产生的，所以也称为阶段产品。如软件的项目计划、需求说明、测试计划、设计文档和源程序，都属于阶段产品的范围。配置管理的中心思想，就是在软件开发的进程中，开发者有权对本阶段的阶段产品进行更改，但一旦阶段产品通过了复审，就应将它交给配置管理人员去控制，任何人(包括编制这一文档的人员)需要对它更改时，都要经过正式的批准手续。在软件工程中，将各个阶段产品的复审时间均称为基线，基线之前可以自由更改，基线之后严格管理，正是这种对软件配置的连续控制与跟踪，保证了软件配置的

完整性与一致性。

8.5 软件评审

在软件生存期每个阶段的工作中都可能引入人为的错误。当出现错误,如果不及时纠正,就会传播到开发的后续阶段中去,并在后续阶段中引出更多的错误。实践证明,提交给测试阶段的程序中包含的错误越多,经过同样时间的测试后,程序中仍然潜伏的错误也越多。所以必须在开发时期的每个阶段,特别是设计阶段结束时要进行严格的技术评审,尽量不让错误传播到下一个阶段。

评审是以提高软件质量为目的的技术活动。缺乏质量概念的技术评审只是一种拘于形式的为评审而评审的工作。为使用户满意,软件质量有两个必要条件。

(1) 设计的规格说明要符合用户的要求。

(2) 程序要按照设计规格说明所规定的情况正确执行。

把上述第一个条件称为设计质量,把第二个条件称为程序质量。如图 8-2 所示,优秀的程序质量是构成好的软件质量的必要条件,但不是充分条件。

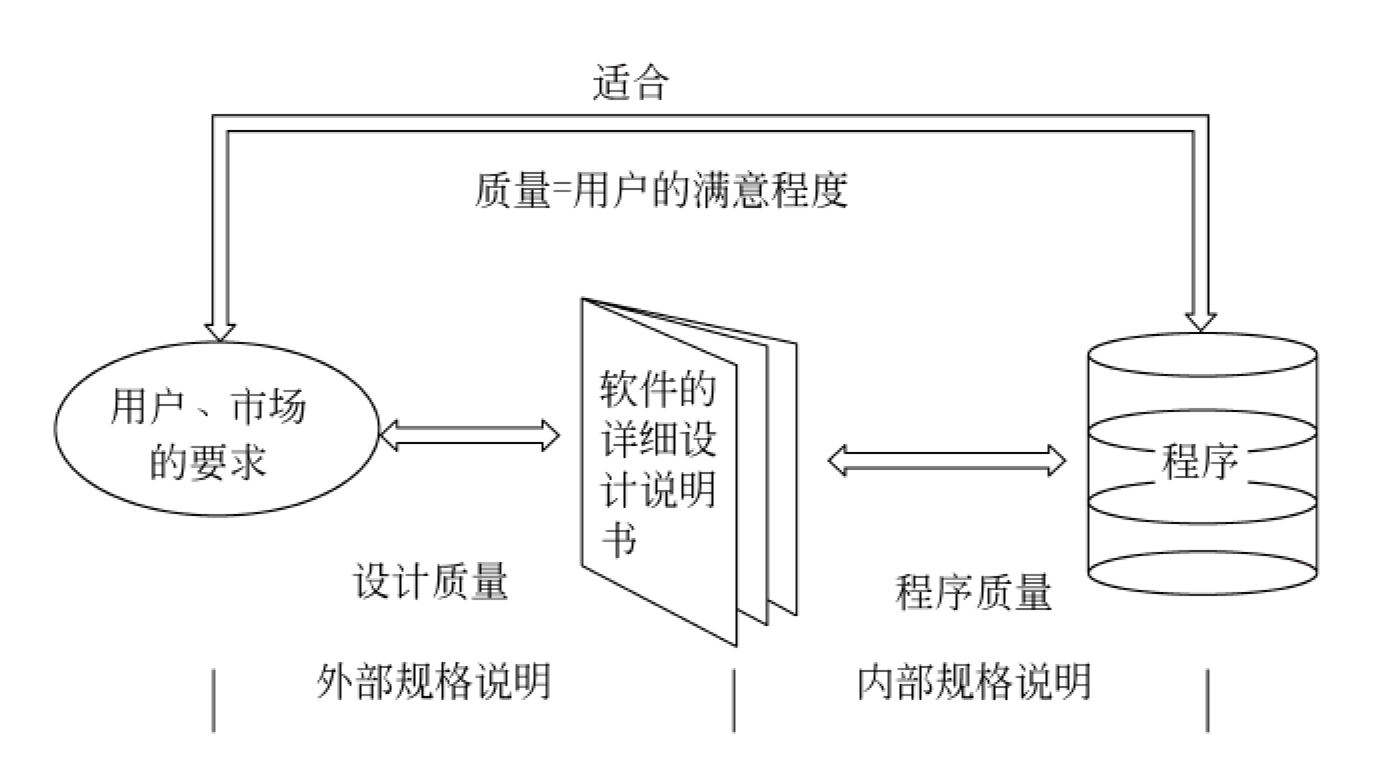


图 8-2 设计质量与程序质量

与上述质量的观点相对应,软件的规格说明可以分为外部规格说明和内部规格说明。外部规格说明是从用户角度来看的规格,包括硬件/软件系统设计(在分析阶段进行)、功能设计(在需求分析阶段与概要设计阶段进行),而内部规格说明是为了实现外部规格的更详细的规格,即程序模块结构与模块加工的设计(在概要设计与详细设计阶段进行)。因此,内部规格说明是从开发者角度来看的规格说明。将上述两个概念联系起来,则可以说明设计质量是由外部规格说明决定的,程序质量是由内部规格说明决定的。下面讨论针对外部规格说明进行的设计评审。

8.5.1 设计质量的评审内容

设计质量的评审对象是在需求分析阶段产生的软件需求规格说明、数据要求规格说明,在软件概要设计阶段产生的软件概要设计说明等。归纳起来,需要从下述 12 个方面

进行评审。

1. 软件的规格说明

(1) 在用户所要求的规格说明(项目合同、用户手册等)中,用户要求是否明确。

(2) 在设计规格说明中各设计结果是否都与用户协商过,并已达成一致意见。

(3) 是否调查了同行业其他单位的相同软件,并进行了比较。

(4) 是否充分分析了正在使用的类似软件,其结果在需求规格说明中是否论述。

(5) 需求规格说明是否得到了用户或用户上级机关的批准。

(6) 上述需求规格说明与软件的概要设计规格说明是否一致。

(7) 总体设计思想和设计方针是否明确。

2. 可靠性

引起系统失效的主要原因有:输入异常(错误或超载等)、硬件失效及软件失效。可靠性措施应能避免这些失效的原因发生,并且一旦发生时,应能及时采取代替手段或恢复手段。因此,评审内容如下。

(1) 要点:用户是否已明确对可靠性的要求和特点,与同类的现有系统是否做过比较,作为比较结果,系统的可靠性要求是否合适,是否充分地分析过同类的现有系统产生失效的数据,是否接受了其经验教训等。

(2) 针对输入错误的处理功能:对输入组合的错误是否能够检查、报错,并提供再输入的手段,能否对正确输入提供帮助,输入值的范围规定是否明确,是否有对范围进行检查的功能,对于上述各种出错情况是否给出明确的出错信息等。

(3) 针对操作错误的处理功能:能否检查操作顺序的错误,能否检查媒体设置的错误,对于以上错误的出错信息是否明确。

(4) 针对硬件失效的处理功能:在硬件失效发生时能否检测故障,是否有再启动功能和自动恢复功能,在硬件失效发生时是否可以缩小使用范围,此时的切换和再配置的单位是否明确,对运行时的功能范围和使用方法是否明确,对硬件失效的恢复方法和顺序是否适当,恢复所需要的时间长短,恢复时的操作是否简便,是否采用冗余性来提高可靠性等。

(5) 针对软件失效的处理功能:是否能自动地在早期检测由于软件故障而导致的混乱和逻辑错误,是否有对发生失效的软件部分自动再连接和再启动功能等。

(6) 该软件是否按照外部规格说明进行工作:是否按照外部规格说明进行工作,对于是否按照外部规格说明进行工作做过充分的测试等。

3. 保密措施实现

(1) 是否有对系统使用资格进行检查的功能。

(2) 是否有对特定数据的使用资格(读入、写出、更新、生成)进行检查的功能。

(3) 是否有对特殊功能的使用资格(如命令的使用限制)进行检查的功能。

(4) 是否有记录以上所列使用情况的功能。

(5) 在查出有违反使用资格情况后,能否向系统管理人员报告有关信息。

(6) 是否有对系统内重要数据加密的功能。

4. 操作特性实施

(1) 操作命令和操作信息的恰当性：在全部操作顺序已明确规定之后，与之相应的命令和信息是否明确了，特别是对系统的启动和终止操作是否明确，系统的中断、再启动、状态监视等控制功能是否明确，系统的状态改变和配置改变的功能是否明确，命令形式是否标准化了，命令的种类是否太多，命令的打入次数是否太多，信息的形式是否标准化了，特别是当异常情况发生时，是否能给出报告信息，带有应答请求的信息所要求的响应方式是否明确，是否已标准化，是否全部命令和信息都能记录，这些命令和信息是否有硬备份。

(2) 输入数据和输入控制语句的恰当性：输入形式是否标准化，输入的描述量是否达到最小输入操作的命令字和参数的表示方式是否易于理解，软件功能和输入控制语句的对应关系是否明确，对输入的默认值和属性等是否做了规定，有关输入形式、输入值和属性的范围等规定是否明确，有关输入的组合和输入顺序的规定是否明确，对违反上述规定的情况是否能检测到，是否有明确的提示信息，上述的规定与现有的或其他的软件的规定是否相同，是否还有其他的输入方法。

(3) 输出数据的恰当性：输出形式是否标准化，标题信息是否合适，有否写上题目名、输出日期等必要的信息，所给出的题目名对不同的输出能否明确地区分开，输出数据的种类明确后，用户是否可以选择自己所需要的种类，输出的单位对用户是否适当，当按成组的方式进行输出时，每个组的划分是否明确，出错信息和输出信息的区别是否明确，是否有输出的代替手段，例如用磁带输出代替行式打印机等。

(4) 应答时间的恰当性：在交互系统或联机系统中的应答时间是否合适。

5. 性能实现

计算机系统的性能由应答时间和处理能力两方面来决定。应答时间对于终端用户的操作很重要，而处理能力对于决定计算机硬件资源的需要量同样重要。一般来说，性能设计是需要考虑多方面因素的复杂工作，因此往往考虑得不很周全，首先应明确规定性能的目标值。

(1) 目标值的恰当性：是否明确规定了目标值，目标值应包括应答时间和各个部分的资源开销，如动态处理的步数、主存储器的使用量、辅助存储器的使用量、对各种外设的输入输出次数、通道的负载情况、线路的负载情况、从发生失效到系统再启动而作业再开始时所需要的恢复时间等。用户是否了解这些目标值，性能目标值是否超过了现有的同类系统，性能的目标值是否考虑到了负载的经常变动情况。

(2) 性能目标设定条件的恰当性：硬件条件，包括机器的名称、型号、系统的构成(即机器的数量及连接情况)；机器的性能，包括指令的平均执行时间、通道传输速度、输入输出速度、主存储器容量、辅助存储器容量、线路的速度等；网络的构成情况；传输控制方式；软件条件，包括所使用的操作系统、有关的其他软件；输入数据的条件，包括平均数据量、最大数据量(考虑日、月、年的最大数据量)、每日的总数据量、数据的种类及其比率；以及操作时间及思考时间。

(3) 明确性能的评价方法：是否考虑到了性能评价用的设备和工具等的准备情况。

6. 可修改性

(1) 检测故障的功能：包括检测由于故障而导致失效的检测功能；检查出故障时的提示功能。

(2) 获取分析数据的功能：包括在故障检测时自动获取有关数据的功能；主存储器数据的转储功能；辅助存储器数据的转储功能；自动记录该软件的工作过程的自动记录功能，即按时间顺序记录该软件的主要工作事件。

(3) 区分问题根源的方法：包括对故障数据进行交互式分析时的支持功能；是否明确失效现象与功能之间、功能与模块之间、功能与模块内程序逻辑之间的对应关系。

(4) 故障修正的方法(包括修改程序库的功能)：能否以模块为单位进行修正，修改所涉及的范围是否明确，修改所涉及的范围是否限制在最小范围，对包括由于修改而影响的范围在内的再测试(回归测试)范围是否明确。

7. 可扩充性

(1) 对扩充问题的考虑：是否掌握了用户的长远系统扩充计划，考虑标准软件时，是否对市场的长期动向进行过分析，是否有与用户约定好的功能扩充计划，是否考虑了硬件的扩充情况，是否能做相应处理，是否有引入新机种的计划，是否有修改结构的计划。

(2) 模块化：功能与模块的对应关系是否明确，模块的大小是否适当，即是否在标准以内，模块之间的关系是否明确。

(3) 模块的通用性：每个模块是否仅是单一功能，是否依赖于特定的硬件，是否依赖于特定的操作系统，对数据量是否有限制，对数据的取值范围是否有限制，模块中使用的常数是否被指定为参数，能否通过指定参数来修改，对所需的存储容量是否有限制。

8. 互换性

互换性是指当软件功能扩充了之后，其已有功能还能照原样使用的特性。特别要注意互换性与可移植性的区别，两者的比较如图 8-3 所示。互换性是指扩充了软件功能，但不影响(不改变)已有软件运行环境的情况；可移植性是指软件运行环境改变时，可不改变软件的规格而能照原样工作的特性。互换性的评审项目如下。

(1) 评审由于软件扩充给运行环境造成的影响：硬件互换性的评价包括 CPU、主存储器、外部输入输出设备、终端等的接口；这些硬件的最大、最小连接数及有关连接方法的互换性。文件互换性的评价包括文件的形式和记录的形式和文件、记录的存取方式。用户界面的互换性，如输入数据与输入控制语句的形式、命令的形式、输出表的形式及输出信息的形式和有关其他输入输出操作的互换性。以及与其他软件接口的互换性，包括程序设计语言、程序库、与操作系统的接口等。

(2) 对可扩充性的考虑。

(3) 软件结构上的稳定性：如果一个软件由多个模块构成，在改变运行环境时，仅修改或更换因环境不同而受影响的那些模块就可达到互换性。图 8-4 就是表示这种想法的示意图。从图中可以看到，当应用部分的功能扩充时，由于中间有与运行环境的接口，所以通过改变这种接口部分，就可保持软件的互换性。

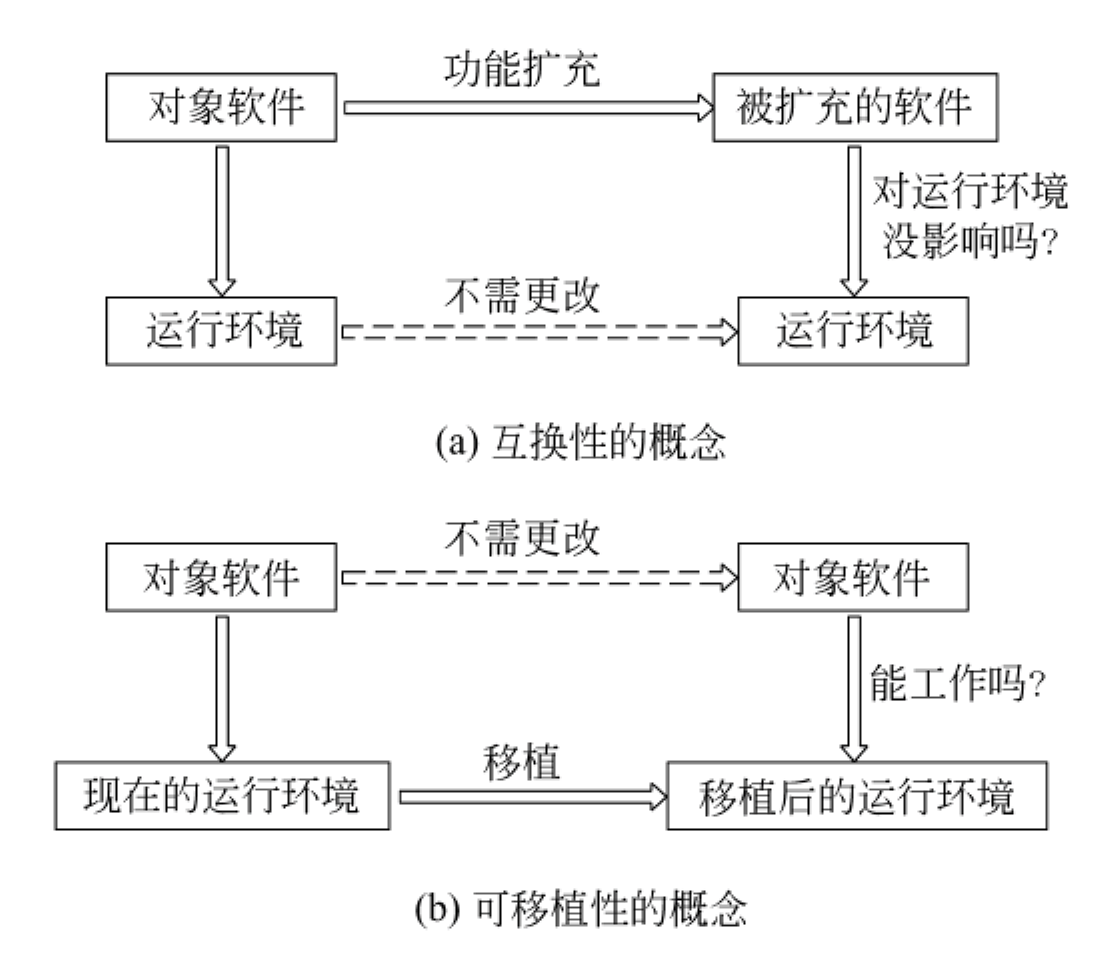


图 8-3　互换性和可移植性的区别

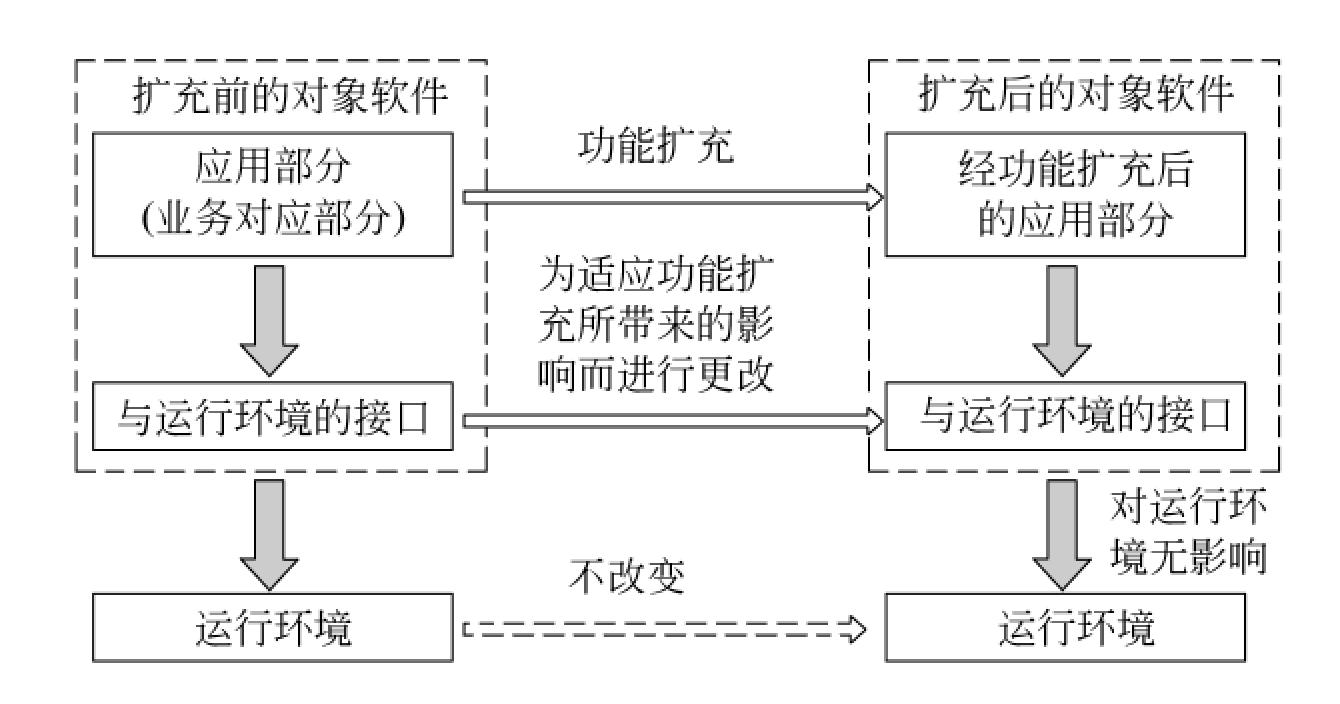


图 8-4　通过接口部分模块化来维持互换性

9. 可移植性

可移植性是指当把软件移植到不同的运行环境时，不需改变其规格就能照原样工作的特性，如图 8-3 所示。另外，即使移植后不能照原样工作，但稍加修改就能正常运行，这也属于可移植性。从图中可以看到，可移植性与互换性是不同的概念。进一步说，可移植性是互换性反过来的概念，所以规定的可移植性的条件中有不少与互换性是共同的。

(1) 其规格是否不易受运行环境影响，例如，为不受 CPU 和操作系统影响，是否用标准的高级程序设计语言书写，输入输出接口是否为逻辑接口，能否不受硬件的影响，文件形式是否采用标准的形式，用户接口是否成为逻辑接口，是否减少了对硬件及操作系统等固有因素的依赖。

(2) 软件接口是否划分出应用部分及与其他部分的接口部分：由于可移植性与互换性的概念相反，因此按照与图 8-3 和图 8-4 表示的想法，反过来去做，就可以保持可移植性。接口部分包括与硬件的接口部分、与文件的接口部分、与用户的接口部分及与其他软件的接口部分。

(3) 向新的运行环境的可移植性：检查在受到运行环境的影响时，只替换②中所列

的接口部分，是否就能做到向新的运行环境的移植。

（4）软件的大小与划分的恰当程度：检查是否能向多种运行环境移植，该软件的大小是否恰当或者是否能划分成适当大小的部分。

10. 可测试性

测试是保证软件质量的重要手段，为了保证软件在修改或扩充后的正确性，不仅要测试所修改或所扩充的部分是否能按规格执行，而且还应对该软件原有的功能经修改或扩充后是否能按以前的规格正确运行进行测试。可测试性是为保证软件质量，有效地进行充分、全面的测试的特性。评审可测试性的项目如下。

（1）检查测试标准情况：是否规定了标准，根据该标准，能否保持质量，用户是否了解这些测试标准，这些测试标准能否规定具体的测试内容。

（2）是否明确设计规格和测试规格之间的对应关系：是否通过测试文档（测试计划、测试用例设计等）明确规定了测试规格，测试规格与设计规格是否明确对应。

（3）限定变化的影响范围：在软件结构的设计中，要减少由于修改或扩充所引起的变化的影响范围，并限定再测试的范围。这种情况与可扩充性是一致的。

（4）是否容易对故障的主要原因进行分析与修改。

（5）检查测试作业：是否明确规定了测试步骤；是否准备了便于进行测试的测试辅助工具。

11. 复用性

复用性应包含可移植性及功能上通用性等，主要包括以下两方面。

（1）具有充分的可移植性。

（2）功能具有通用性。

12. 互连性

满足互连性的必要条件是与其他软件有共同的接口及该接口部分是模块化的，容易改变的。

（1）与其他软件应有共同接口：与其他软件之间是否有公共的数据定义形式，根据这个数据定义形式，能否定出不同数据间的转换标准，与其他软件之间是否有公共的通信方式的定义形式，根据这个通信方式的定义形式，能否定出不同的通信方式间的转换标准。

（2）与其他软件之间的接口部分应是模块化的。

8.5.2 程序质量的评审内容

程序质量评审着眼于软件本身的结构、与运行环境的接口、变化带来的影响而进行的评审活动。通常它是从开发者的角度进行评审的，直接与开发技术有关。

1. 软件的结构

为了使软件能够满足设计规格说明中的要求，软件的结构本身必须是优秀的。

1）功能结构

在软件的各种结构中，功能结构是用户唯一能见到的结构。因此，功能结构可以说是

联系用户和开发者的规格说明,它在软件的设计中占有极其重要的地位。软件功能的本质是把输入信息变换为输出信息。因此,在讨论软件的功能结构时,必须明确软件的数据结构。需要检查的项目有以下几项。

(1) 数据结构:包括数据名和定义、构成该数据的数据项、数据与数据项间的关系。例如,数据 A 是由数据项 B 和 C 构成时,A 是 B 与 C 的组合数据还是由 B 或 C 中某一方选择出来的数据。

(2) 功能结构:包括功能名和定义、构成该功能的子功能、功能与子功能之间的关系。

(3) 数据结构和功能结构之间的对应关系:包括数据元素与功能元素之间的对应关系、数据结构与功能结构的一致性。

2) 功能的通用性

在软件的功能结构中,某些功能有时可以作为通用功能反复出现多次。从功能便于理解、增强软件的通用性及降低开发的工作量等观点出发,希望尽可能多地使功能通用化。实现功能通用化的最一般的方法是通过提取公用功能来实现通用模块化。而进一步的功能通用化方法就是信息隐蔽或数据抽象。它的基础就是抽象数据类型。在实现功能通用性方面,检查项目如下。

(1) 抽象数据结构:包括抽象数据的名称和定义、抽象数据构成元素的定义。

(2) 抽象功能结构:包括(1)中的抽象数据进行操作的各个功能的一览表、上述各功能的定义及各个功能之间的关系。

3) 模块的层次

模块的层次就是指程序模块结构。由于模块是功能的具体体现,所以模块层次应当根据功能层次来设计。模块层次中保有一部分功能层次,但模块层次并不全与功能层次相同,重要的是应明确模块层次与功能层次之间的关系。为此,要检查以下项目。

(1) 模块层次:模块层次的定义,包括各层次的含义、各层次物理容量的上限;模块的层次结构,包括各模块间的联系、各模块与层次的对应关系。

(2) 与功能层次的对应关系:功能与模块的对应关系;功能层次与模块层次的匹配程度。

4) 模块结构

以上所述的模块层次结构是模块的静态结构,现在要检查模块间的动态结构。模块分为处理模块和数据模块两类。模块间的动态结构也与这些模块分类有关。对这样的模块结构进行检查的项目有以下几项。

(1) 控制流结构:这种结构规定了处理模块与处理模块之间的流程关系。因此,要检查处理模块之间的控制转移关系和控制转移形式(调用方式)。

(2) 数据流结构:这种结构规定了数据模块是如何被处理模块进行加工的流程关系。因此,要检查处理模块与数据模块之间的对应关系和处理模块与数据模块之间的存取关系(建立、删除、查询、提取、修改等)。

(3) 模块结构与功能结构之间的对应关系：包括功能结构与控制流结构的对应关系和功能结构与数据流结构的对应关系。

(4) 每个模块的定义：包括功能、输入与输出数据。

5) 处理过程的结构

处理过程是指模块划分到最底层的那些模块的实现方式，也就是最基本的加工逻辑过程。对它的检查项目有以下几项。

(1) 要求模块的功能结构与实现这些功能的处理过程的结构应明确对应。

(2) 要求控制流应是结构化的。

(3) 数据的结构与控制流之间的对应关系应是明确的，并且可依这种对应关系来明确数据流程的关系。

(4) 用于描述的术语标准化。

2. 与运行环境的接口

运行环境包括硬件、其他软件和用户。与运行环境的接口应设计得较理想，要预见到环境改变，并且一旦要变化时，应尽量限定其变化范围和变化所影响的范围。

主要检查项目如下。

(1) 与其他软件的接口：包括与上层软件的接口，如与操作系统等控制该软件的那些软件的接口；与同层软件的接口，如通过文件连接起来的那些软件的接口；与下层软件的接口，如编译程序与作为其输入的源程序之间的接口。

(2) 与硬件的接口：包括与硬件的接口约定(即根据硬件的使用说明等所做出的规定)和硬件故障时的处理和超载时的处理。

(3) 与用户的接口，包括以下几种。

① 与用户的接口规定，即通过用户手册做出的规定。

② 输入数据的结构，如输入的种类、输入的形式、输入的数据量、输入表达式的容易理解程度、输入方法(输入设备种类)等。

③ 输出数据的结构，如输出的种类、输出的形式、输出的数据量、输出表达式的容易理解程度、输出方法(输出设备种类)等。

④ 异常输入时的处理。

⑤ 超载输入时的处理。

⑥ 用户存取资格的检查。

⑦ 变化的影响范围问题。

随着软件运行环境的变化，软件的规格也跟着不断地变化。在处理变化的过程中，重要的是要明确地掌握变化的影响范围，并且在设计上努力缩小其变化的影响范围。

(4) 运行环境变化时的影响范围。

① 与运行环境的接口：包括与其他软件的接口、与硬件的接口及与用户的接口。这是变化的重要原因，需要注意的是对那些不要求变化的部分的副作用。

② 在每项设计工程规格内的影响：包括功能结构、功能的通用性、模块层次、模块结构及处理过程的结构。这是在每个软件结构范围内的影响。例如，若改变某一功能，则与之相联系的父功能和它的子功能都会受到影响。如果要变化某一模块，则调用该模块的

其他模块都会受到影响。

③ 在设计工程相互间的影响：包括功能结构与功能的通用性、功能结构与模块层次、模块层次与模块结构及模块结构与处理过程的结构。这是指不同种类的软件结构相互间的影响。例如，当改变某一功能时，就会影响到模块的层次及模块结构，这些对模块的处理过程都将产生影响。

8.6　软件质量保证的标准

质量保证系统可以定义为用于实现质量管理的组织结构、责任、规程、过程和资源。ISO 9000 标准以一种能够适用于任何行业的术语描述了质量保证的要素。

为了登记成为 ISO 9000 中包含的质量保证系统模型，一个公司的质量系统和操作应该被第三方审计者仔细检查，查看其与标准的符合性以及操作的有效性。成功登记之后，这一公司将收到由审计者所代表的登记实体颁发的证书。此后每半年进行一次的检查性审计持续地保证该公司的质量系统与标准相符。

1. ISO 对质量保证系统的方法

ISO 9000 质量保证模型将一个企业看作一个互联过程的网络。为了使质量系统符合 ISO 标准，这些过程必须与标准中给出的区域对应，并且必须按照描述进行文档化和实现。对一个过程文档化将有助于组织的理解、控制和改进。正是理解、控制和改进过程网络的机能为设计和实现符合 ISO 的质量系统的组织提供了最大的效益。

ISO 9000 以一般术语描述了一个质量保证系统的要素。这些要素包括用于实现质量计划、质量控制、质量保证和质量改进所需的组织结构、规程、过程和资源。但是 ISO 9000 并不描述一个组织应该如何实现这些质量系统要素。因此，重要的工作在于如何设计和实现一个能够满足标准并适用于公司的产品、服务和文化的质量保证系统。

2. ISO 9001 标准

ISO 9001 是应用于软件工程的质量保证标准。这一标准中包含了高效的质量保证系统必须体现的 20 条需求。因为 ISO 9001 标准适用于所有的工程行业，因此，为了在软件使用的过程中帮助解释该标准，而专门开发了一个 ISO 指南的子集（即 ISO 9000—3）。

由 ISO 9001 描述的 20 条需求所面向的是以下问题。

(1) 管理责任。

(2) 质量系统。

(3) 合同复审。

(4) 设计控制。

(5) 文档和数据控制。

(6) 采购。

(7) 对客户提供的产品的控制。

(8) 产品标识和可跟踪性。

(9) 过程控制。

(10) 审查和测试。
(11) 审查、度量和测试设备的控制。
(12) 审查和测试状态。
(13) 对不符合标准产品的控制。
(14) 改正和预防行动。
(15) 处理、存储、包装、保存和交付。
(16) 质量记录的控制。
(17) 内部质量审计。
(18) 培训。
(19) 服务。
(20) 统计技术。

软件组织为了通过 ISO 9001,就必须针对上述每一条需求建立相关政策和过程,并且显示组织活动的确是按照这些政策和过程进行的。

8.7 软件质量评价

在评价软件质量时,对于软件功能、结构、层次等的描述比较模糊,不能确切地评价软件的质量,更不能作为企业选购软件的依据。在企业引进一款软件的时候,经常会出现以下一些问题,使得企业无法正确地评判软件的质量。

(1) 定制的软件可能难于理解,难于修改,在维护期间,企业的维护费用大幅度增加。

(2) 企业评价软件质量没有恰当的指标,对软件可靠性和功能性指标了解不足。

(3) 软件开发者缺乏历史数据作为指南,所以进度和成本的估算粗略。因为没有切实的生产率指标,没有过去关于软件开发过程的数据,企业无法精确评价开发商的工作质量。

8.7.1 软件质量评价体系

归纳起来,获取软件的途径有 4 种:自行开发,直接外购,外购再二次开发,与软件开发商合作开发。而其中又以合作开发最为普遍,因为这种方式更能满足企业独特的业务流程,更有针对性。美国的 B. W. Boehm 和 R. Brown 提出了三层的评价度量模型:软件质量要素、评价准则、度量。在此基础上,出现了各种软件质量度量技术。G. Mruine 提出了软件质量度量 SQM 技术,并成功应用到了波音公司的软件开发过程中。日本的 NEC 公司也提出了软件质量度量 SQMAT 技术,并且在成本控制和进度安排方面取得了良好的效果。

1. 软件质量要素

软件质量可分解成 6 个要素,这些要素构成了软件的基本特征。

(1) 功能性:软件所实现的功能满足用户需求的程度。功能性反映了所开发的软件满足用户描述的或蕴涵的需求的程度,即用户要求的功能实现程度。

(2) 可靠性：在规定的时间和条件下，软件所能维持其性能水平的程度。可靠性不仅描述了软件满足用户需求正常运行的程度，也描述了在故障发生时软件可以继续运行的程度。

软件可靠性的依据不是软件已经过多少周的测试、调试，而是在可靠性预测模型中，定量地估计出软件中每千行代码存在多少个错误没有被消除。更进一步，通过软件质量测量，用户知道该软件在使用中的平均失效前工作时间(Mean Time To Failures，MTTF)和平均失效间隔时间(Mean Time Between Failures，MTBF)。

(3) 易用性：是指对于一个软件，用户学习、操作、准备输入和理解输出时，所做努力的程度。易用性描述了软件产品的友好性，即用户在使用本软件时是否方便。

(4) 效率：在指定的条件下，用软件实现某种功能所需资源(例如时间)的有效程度。该要素描述了在完成功能要求的过程中，资源的使用程度。

(5) 可维修性：可维修性描述了在用户需求改变或软件环境发生变化时，对软件系统进行相应修改的容易程度。在软件运行过程中，为了满足用户需求改变、运行环境改变或软件错误发生时，进行相应修改时要做的工作的程度描述。一个易于维护的软件系统也是一个易理解、易测试和易修改的软件，以便纠正或增加新的功能，并且可以允许软件在不同的环境中进行操作。

(6) 可移植性：从一个计算机系统或环境转移到另一个计算机系统或环境的难易程度。

2. 评价准则

评价度量模型的第二层是评价准则，一共包括22个要素：精确性、健壮性、安全性以及通信有效性、处理有效性、设备有效性、可操作性、培训性、完备性、一致性、可追踪性、可见性、硬件系统无关性、软件系统无关性、可扩充性、公用性、模块性、清晰性、自描述性、简单性、结构性、产品文件完备性。其中精确性、健壮性、安全性三条尤其重要。

(1) 精确性：在计算和输出时所需精度的软件属性。

(2) 健壮性：在发生意外时，能继续执行和恢复系统的软件属性。

(3) 安全性：防止软件受到意外或蓄意的存取、使用、修改、毁坏或泄密的软件属性。

评价准则的一定组合将反映某一软件质量要素，软件质量要素与评价准则间的关系如图8-5所示。

3. 度量

评价度量模型的第三层是度量，根据软件工程的7个阶段：需求分析、概要设计、详细设计、编码实现、测试、运行和维护，制定出针对每一个阶段的问卷表，以此实现软件开发过程的质量控制。例如在需求分析、概要设计、详细设计及其实现阶段，主要评价软件需求是否完备，设计是否完全反映了需求，以及编码是否简洁、清晰。

软件质量很大程度上取决于用户的参与程度，所以不管是定制，还是外购软件后的二次开发，让客户了解并监控软件开发过程每一个环节的进展情况，对产品水平提高相当重要。

这里需要说明几点。

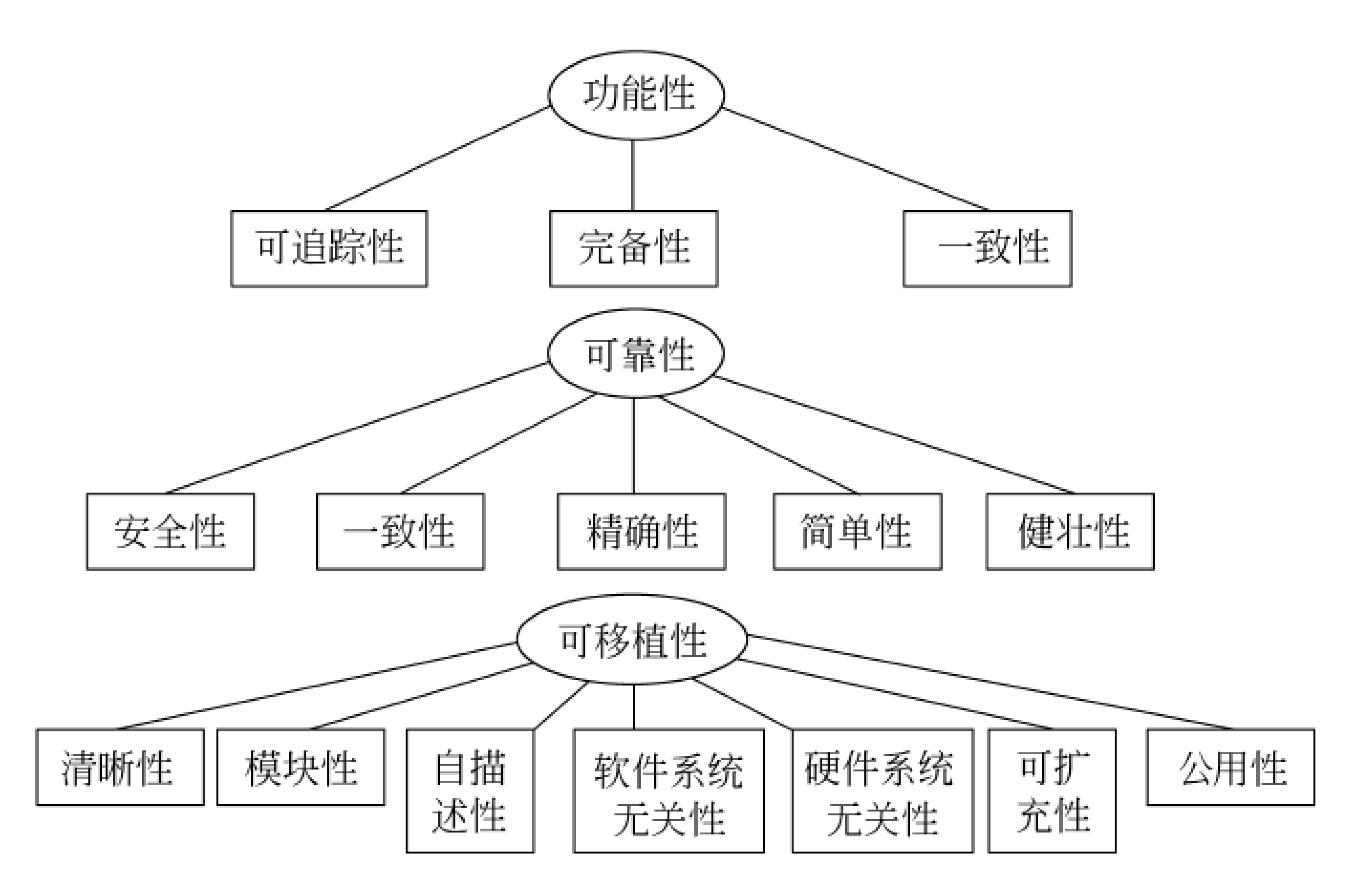


图 8-5 软件质量要素与评价准则间的关系

(1) 不同类型的软件和不同规模的软件在质量要求、评价准则、度量问题等方面的侧重点有所不同。例如系统软件、控制软件、管理软件、CAD 软件、教育软件、网络软件这些针对不同类型客户的软件度量需要考虑的要素也不同。

(2) 严格对软件质量各阶段进行度量。度量的目的是以此控制成本、进度,改善软件开发的效率和质量。

(3) 结合本企业的实际软件经验。例如企业选择软件供应商、开发商,需要考察该公司是否建立起自己的软件质量度量和评价数据,数据库中是否存有与本企业所在行业相关的软件,是否具有相关的开发经验。

8.7.2 软件质量评价标准

1. 影响软件质量的因素

影响软件质量的因素分为以下 3 类,简述如下。

1) 产品运行

主要内容如下。

(1) 正确性:能按需要工作。

(2) 健壮性:对意外环境能适当地响应。

(3) 效率:完成预定功能时需要较少的计算机资源。

(4) 完整性:是完整的。

(5) 可用性:能使用。

(6) 风险性:能按预定计划完成。

2) 产品修改

主要内容如下。

(1) 可理解性:能理解。

(2) 可维修性:能修复。

(3) 灵活性：能改变。

(4) 可测试性：能测试。

3) 产品转移

主要内容如下。

(1) 可移植性：能在另一台机器上使用。

(2) 可重用性：能再用它的某些部分。

(3) 可运行性：能把它和另一个系统结合。

2. 全面质量控制的要点

1) 实行工程化开发

构建软件产品是一项系统工程，必须建立严格的工程控制方法，要求开发组的每一个人都要遵守工程规范，规范由一系列活动组成的方法体系。按规范工作可以较合理地达到目标。

2) 实行阶段性冻结与改动控制

软件产品具有生存周期，这就为划分项目阶段提供了参考。一个大项目可分成若干阶段，每个阶段有各自的任务和成果。这样一方面便于管理和控制工程进度，另一方面可以增强开发人员和用户的信心。每个阶段末的成果作为下一阶段开发的基础。阶段末的成果修改要经过一定的审批程序，并且涉及项目计划的调整。

3) 里程碑式审查与版本控制

里程碑式审查就是在信息软件生存周期每个阶段结束之前，都正式使用结束标准对该阶段的成果进行严格的技术审查，如果发现问题，就可以及时在该阶段内解决。版本控制是保证项目小组顺利工作的重要技术。版本控制是指通过给文档和程序文件编上版本号，记录每次的修改信息，使项目组的所有成员都了解文档和程序的修改过程。版本控制技术又称为软件配置管理，常用的软件工具有 PVCS 和 Microsoft Visual Source Safe 等。

4) 实行面向用户参与的原型演化

在每个阶段的后期，快速建立反映该阶段成果的原型系统，通过原型系统与用户交互，及时得到反馈信息，验证该阶段的成果并及时纠正错误，这一技术被称为“原型演化”。原型演化技术需要先进的 CASE 工具的支持。

5) 尽量采用面向对象和基于构件的方法

面向对象的方法强调类、封装和继承，能提高软件的可重用性，将错误和缺陷局部化，同时还有利于用户的参与，这些有助于提高软件系统的质量。

基于构件的开发又称为即插即用编程方法，是从计算机硬件设计中吸收过来的优秀方法。这种编程方法是将编制好的“构件”插入已做好的框架中，从而形成一个大型软件。构件是可重用的软件部分，构件既可以自己开发，也可以使用其他项目的开发成果，或者直接向软件供应商购买。当发现某个构件不符合要求时，可对其进行修改而不会影响其他构件，也不会影响系统功能的实现和测试。

6) 全面测试

要采用适当的手段，对系统调查、系统分析、系统设计、实现和文档进行全面测试。

7）引入外部监理与审计

要重视软件系统的项目管理，特别是项目人力资源的管理，因为项目成员的素质和能力以及积极性是项目成败的关键。同时还要重视第三方监理和审计的引入，通过第三方的审查和监督来确保项目质量。

8.8 软件质量框架

软件质量框架是提出一个评价的原型，帮助提高软件质量。

8.8.1 高质量软件的特性

高质量的软件通常具有以下特性。

（1）满足用户的需求。这是软件质量的第一评判标准，一个软件如果不能够满足用户的需要，设计得再好，采用的技术再先进，不能被客户认可，也是没有任何意义的。

（2）合理安排进度、成本、功能关系。如何在特定的时间内，以特定的成本，开发出特定功能的软件，高质量的软件的开发过程中，一定要客观地对待这3个因素，并通过有效的计划、管理、控制，使得三者之间达成平衡，才能保证产出的最大化。

（3）具备扩展性和灵活性，能够适应一定程度的需求变化。一个质量优秀的软件，应该能够在一定程度上适应社会需求和企业需要的变化，从而保持软件的稳定。

（4）能够有效地处理特殊情况。高质量的软件应该可以及时处理各种特殊情况，例如断电、系统异常等。一个软件如果具有足够的鲁棒性，就能够承受各种非法情况的冲击。

（5）能够可持续地发展。一个优秀的软件在开发完成后，可以形成知识沉淀，为软件开发的长期发展贡献力量。

8.8.2 软件质量框架的组成

软件质量框架的构建是以敏捷方法论为基础的，并将先进的软件开发技术融入其中。软件质量框架主要由前提、价值观、结构和实践组成。

1. 前提

（1）系统平台前提：由于软件质量框架的实践将会涉及具体的技术和代码，所以首先为软件质量框架定义软件架构的平台，如J2EE或是.NET平台，再采用对象分析技术。

（2）组织前提：要考虑评估应用软件质量框架需要多少的投入，对目前的开发过程有多大的收益。一般来说，组织的规模越大、其开发过程和产品越复杂，就越适合采用软件质量框架。

（3）方法学前提：在敏捷方法学中，对规则和秩序有两种不同的观点，一种是强调规则和秩序，对代码都有要求；另一种以自适应软件开发为代表，它不要求程序员的具体行为。一般软件质量框架采用第一种观点，要求组织中存在严谨的规则和秩序。

2. 价值观

(1) 明确：对软件的管理必须是明确具体的，任何模糊的指令都可能导致软件开发中的错误。例如，可以把需求文档写得非常具体，但是需要付出制作和维护的代价，所以指令越明确，成本就越高。

(2) 容错：软件开发错误是无法避免的。所以，软件质量框架中是允许犯错的。如果强制规定不能出错，则可能会引发其他的问题，例如隐瞒错误，或为了隐瞒错误而导致的额外成本。所以正确的态度是允许发生错误，并建立一套监测、管理、反馈、修改错误的体制。

(3) 规范：在软件质量框架中强调规范，并使用规范来推动框架的运作。

(4) 测试：测试是保证软件质量的必由之路。测试要尽可能地多，尽可能地频繁，测试结果要尽可能快地反馈。测试是综合性的，软件开发过程中的所有过程，都需要伴随着相应的测试。

软件质量框架是优秀软件开发思想的一个应用，是对软件开发过程的有效管理实践。它以敏捷方法论为基础，并将先进的软件开发技术融入其中。

3. 结构

图 8-6 描述了软件质量框架的结构，它主要由技术架构、管理架构、支撑框架和业务架构组成。软件质量框架不仅仅关注单个开发人员的效率，更注重整个开发团队的整体效率。因此，管理架构在框架中定义了一套软件管理的方法，能够对开发人员及他们的工作进行管理。在管理架构的基础上，软件质量框架又提供了一个技术架构，其目的是明确定义如何应用组织中涉及的软件技术，以及管理软件技术的方法。技术架构是具体的代码，比方法学更加具体，更容易理解。所以技术架构有利于技术积累和为管理架构服务。

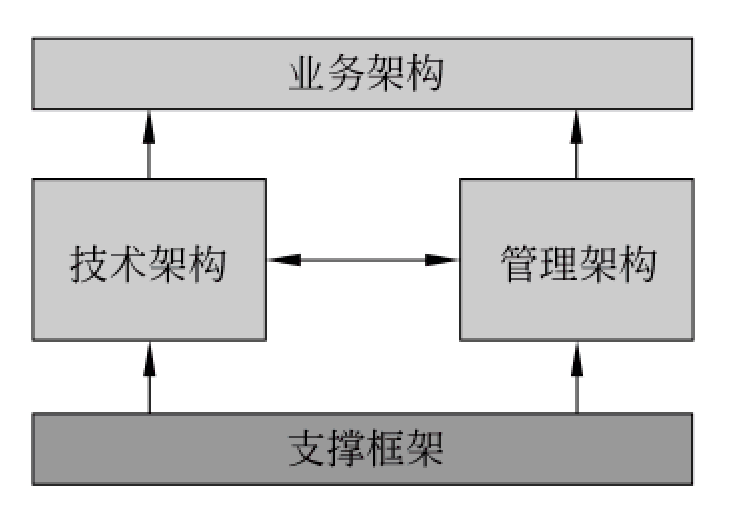


图 8-6　软件质量框架的结构

技术架构和管理架构的下一层是支撑框架。支撑框架包括代码、组件、文档，目的是为技术架构和管理架构提供底层的支持。

处于结构最顶层的是业务架构。因为不同的软件组织的业务不同，所以业务架构也不同。业务架构的目的是对业务进行建模和抽象，提取出可重用的部分，以提高软件开发的效率。

4. 实践

实践来源于软件开发界中的一些新思路和新理论，不论是在成本控制上，还是在质量的改进上，都促进了软件质量的提高。

(1) 日创建：工作流程能够指导软件开发的进行。这个流程应当是具体的、可操作的。日创建实践提出了一种软件开发过程精细管理的方法，是量化软件管理的基础。有了日创建，计划的制定和进度的监控将变得更为容易。

(2) 测试驱动开发：测试驱动开发保证了测试的投入能够带来软件质量的有效提

升。测试不是一个完整的方法论,可以和任何一种开发流程融合。测试驱动开发不但能够改善测试效果,还能够改进软件的设计。

(3) 建立核心框架:传统的知识积累的方法是记录文档,但是文档容易产生歧义,开发人员往往也不愿意去阅读和理解文档。框架提供的是一种综合的手段,包括文档、模型和代码,更容易理解。更重要的是,开发人员应该在日常的工作中就使用框架,这使得他们对框架中的知识非常熟悉,并根据工作的需要来改进框架。

(4) 面向组件编程:面向组件编程采用更加细密的划分方式,替代了传统的以功能块为单位的粗粒度划分方式,并以服务作为组件之间相互依赖的契约,不但定义了组件和组件之间的关系,也规定了组件开发者、组件使用者、组件测试者的权利和义务,从而能够进行软件开发工作的分配、管理等工作。

以上的内容实践都能够对软件质量的改进起积极的作用。此外,它们为软件质量框架结构的实现提供了一个明确的实现方式。从软件结构的角度来看,日创建和测试驱动开发似乎偏向于管理架构,而建立核心框架和面向组件编程则偏向于技术架构。事实上,它们既包含技术架构,也包含管理架构,彼此之间也有相互关联。

8.9 软件开发质量的定量描述

软件的质量与需求描述、建模设计、编码以及软件测试密切相关。掌握定量评估软件开发中产生的分析及设计模型、源代码和测试用例质量的方法是十分重要的。为了实时的质量评估,应该采用技术度量来客观地评估质量。

软件需求是度量软件质量的基础。在定量监理实践中,通常需要使用一种称之为尺度度量的方法,这种方法适用于能够直接度量的特性,例如,出错率定义为错误数、KLOC、单位时间等。因而,为了实现质量控制而应建立的定量数据如下所述。

(1) 数据应具有无歧义性、完全性、正确性、可理解性、可验证性、内部和外部一致性、可完成性、简洁性、可追踪性、可修改性、精确性和可复用性。这些数据可以用来评价分析模型和相应的需求规约质量的特征。

(2) 产品发布前清除的缺陷数在总缺陷数中所占的百分比,有助于评估产品的质量。

(3) 按缺陷的程度来统计出平均修复时间,将有助于规划纠正缺陷的工作。

(4) 利用测试的统计数据,估算可维护性、可靠性、可用性和原有故障总数等。有助于评估应用软件的稳定程度和可能产生的失败。

8.9.1 基本的定量估算

设 F 为用功能点描述的软件规模;D_1 为在开发过程(提交之前)中发现的所有缺陷数;D_2 为提交后发现的缺陷数;D 为发现的总缺陷数。

因此,$D=D_1+D_2$

对于一个应用软件项目,则有如下计算方程式(可以从不同的角度估算软件的质量):

质量$=D_2/F$;

缺陷注入率$=D/F$；

整体缺陷清除率$=D_1/D$。

假如人力资源管理软件的功能点 F 为 366，而在开发过程中发现了 15 个错误，提交后又发现了 4 个错误，则：

$D_1=15, D_2=4$；

$D=D_1+D_2=15+4=19$；

质量（每功能点的缺陷数）$=D_2/F=4/366\approx0.0109$；

缺陷注入率$=D/F=19/366\approx0.05191$；

整体缺陷清除率$=D_1/D=15/19\approx0.7895$。

8.9.2 软件需求的估算

假设在一个规约中有 n_r 个需求，所以：

$$n_r = n_f + n_{nf}$$

其中，n_f是功能需求的数目，n_{nf}是非功能需求数目（例如性能）。

为了确定需求的确定性（无二义性），复审者对每个需求解释的一致性的度量方法为：

$$Q_1 = nSum/n_r$$

其中，Q_1 表示需求的确定性，$nSum$ 是所有复审者都有相同解释的需求数目。当需求的模糊性越低时，Q_1 的值越接近 1。

在人力资源管理软件的例子中，假设图形显示功能模块的功能性需求是 10 个，非功能性需求（响应速度和分辨率）是 2 个，所有复审者都有相同解释的需求数目是 11 个，则：

$$Q_1 = 11/12 \approx 0.916667$$

而功能需求的完整性 Q_2 则可以通过计算以下比率获得：

$$Q_2 = n_u/(n_i \times n_s)$$

其中，n_u 是唯一功能需求的数目，n_i 是由规约定义或包含的输入（刺激）的个数，n_s 是被表示的状态的个数。

Q_2 只是一个系统所表示的必需的功能百分比，但是没有考虑非功能需求。为了把这些非功能需求结合到整体度量中，必须考虑已有需求已经被确认的程度，可以用 Q_3 来表示：

$$Q_3 = n_c/(n_c + n_{nv})$$

其中，n_c 是已经确认为正确的需求的个数，n_{nv}是未被确认的需求的个数。

在所列举的软件中，假设数据库管理功能模块的唯一功能需求是 10 个，由规约定义或包含的输入个数也是 10 个，表示的状态的个数是 1 个，已经被确认的需求是 8 个，未被确认的需求是 2 个，则：

$$Q_2 = 10/(10 \times 1) = 1.0$$

$$Q_3 = 8/(8+2) = 0.8$$

8.9.3 估算验收测试阶段预期发现的缺陷数

如果使用类似项目的数据，就可以估计当前项目在验收测试时发现的缺陷数，它等于

在类似项目的验收测试阶段发现的缺陷数和这个项目估计的工作量与实际总工作量比率的乘积。用如下公式表示：

验收测试缺陷的估计 = 验收测试缺陷数 × 工作量估计 / 实际工作量

在人力资源管理软件中，若以前有一个相似的学籍管理软件，在验收测试的时候发现了 12 个缺陷，本项目估算的工作量是 66 人/月，实际的工作量是 82 人/月，则在验收测试时可能出现的缺陷是：

验收测试缺陷的估计＝12×66/82≈10

8.9.4 维护活动设计的度量

IEEE Std. 982.1—1988[IEE94]提出了软件成熟度指标(SMI)，它提供了对软件产品的稳定性的指示(基于为每一个产品的发布而做的变动)，以下信息可以确定。

MT＝当前发布中的模块数；

F_c＝当前发布中已经变动的模块数；

F_a＝当前发布中已经增加的模块数；

F_d＝当前发布中已删除的前一发布中的模块数。

软件成熟度指标计算公式如下：

$$SMI = [MT - (F_a + F_c + F_d)]/MT$$

当 SMI 接近 1.0 的时候，产品开始稳定。SMI 也可以用作计划软件维护活动的度量。产生一个软件产品的发布的平均时间可以和 SMI 关联起来，并且也可以开发一个维护工作量的经验模型。

在人力资源管理软件的例子中，若目前的软件是 2.0 版，当前发布的模块数是 32 个，当前发布中已经变动的模块数是 8 个，当前发布中已经增加的模块数是 2 个，当前发布中已删除的前一发布中的模块数是 1 个，则：

$$SMI = (32 - 8 - 2 - 1)/32 \approx 0.656$$

从结果可以看出，目前的情况离产品稳定还有相当的距离。

8.9.5 软件可用性的计算

软件可用性是指在某个给定时间点上程序能够按照需求执行的概率。其定义为：

$$可用性 = MTTF/(MTTF + MTTR) \times 100\%$$

其中，$MTTF$ 是平均失败时间，$MTTR$ 是平均修复时间。

在人力资源管理软件的例子中，若软件在 6 个月内失败一次，每次恢复平均需要 20 分钟(恢复时间为排除故障或系统重新启动所用的时间)，可用性是：

$$6\ 个月\ /(6\ 个月 + 20\ 分钟) \times 100\% \approx 99.99\%$$

提高系统的可用性基本上有两种方法：即增加 $MTTF$ 或减少 $MTTR$。而增加 $MTTF$ 还要求增加系统的可靠性。

8.9.6 利用植入故障法估算程序中原有故障总数 E_N

可以采用捕获/再捕获抽样法来估算程序中原有故障总数。设 N_s 是在测试前人为

地向程序中植入的故障数(称播种故障)，n_s 是经过一段时间测试后发现的播种故障的数目，n 是在测试中又发现的程序原有故障数。

假设测试用例发现植入故障和原有故障的能力相同，则程序中原有故障总数 $N(=ET)$ 估算值为：$E_N=(N_s/n_s)\times n$。

例如，在人力资源管理软件的测试过程中，播种故障数是 8 个，经过一段时间的测试后发现的播种故障数是 4 个，在测试中又发现原有的故障数是 2 个，则程序中原有的故障总数 E_N 是：

$$E_N = (8/4) \times 2 = 4$$

小　　结

本章主要介绍了软件质量和软件质量保证的定义与概念、软件质量保证活动、软件质量保证的标准、软件质量评价、软件质量框架、软件开发质量等内容。通过这些内容的学习，可以使学生对软件质量、软件质量保证及软件质量评价等系列问题有深刻的理解和掌握，为构建高质量的软件打下坚实基础。

习　题　8

1. 简述软件质量保证策略。
2. 软件规格说明包括哪些内容？
3. 假如库房管理软件的功能点 F 为 566，而在开发过程中发现了 20 个错误，提交后又发现了 6 个错误，则计算质量(每功能点的缺陷数)、缺陷注入率、整体缺陷清除率各为多少。
4. 例如，在人力资源管理软件的测试过程中，播种故障数是 18 个，经过一段时间的测试后发现的播种故障数是 14 个，在测试中又发现原有的故障数是 4 个，则程序中原有的故障总数 E_N 为多少？

第 9 章　软件测试工具

学习要点：

- 测试工具的作用。
- 测试工具的分类。
- 典型的测试工具。
- 测试工具的选择。

随着软件系统规模增大和广泛应用，测试的重要性逐步增加，使用测试工具已经成了普遍的趋势。目前专门用于测试的工具比较多，这些测试工具可分为白盒测试工具、黑盒测试工具、性能测试工具，另外还有用于测试管理的工具，如测试流程管理、缺陷跟踪管理、测试用例管理等工具。

为了在软件测试时充分发挥测试工具的功能，就必须非常熟悉测试工具的各种功能和操作方法，并可以在不同的阶段使用恰当的测试工具。另外，在使用自动化测试工具的时候，测试人员还得编写测试程序。而不同的测试工具编写代码所用的脚本语言也不相同。这些都需要测试人员付出一定的劳动。

测试工具的应用提高了测试的质量与效率。但是，在实际的测试过程中，并不是所有的测试工具都适用。因此本章将简要地介绍目前常用的测试工具，有利于用户选择和应用。

9.1　测试工具的作用

工欲善其事，必先利其器。使用测试工具的目的是提高软件测试的质量和工作效率。在测试过程中应用测试工具的优势如下所述。

1. 找出潜在的错误

利用测试工具所找到的错误通常是人工测试无法察觉的潜在风险。例如当打开一个窗口后再将它关闭时，系统的内存使用量也由于窗口的关闭而减少。但是利用相应的测试工具就可以探测到类似的风险。使用测试工具不仅可以找到潜在的缺陷，而且还可以帮助测试人员进行问题的诊断，即对错误进行定位。测试人员可以借助不同的测试工具将发现的问题范围逐渐缩小直至找到错误的根源。

2. 有助于诊断错误

使用测试工具有助于问题的诊断,诊断即指定位。当有问题发生时,如果不能确定问题的所在,就必须搜集更多的信息,或者借助软件测试工具的帮助,将问题范围逐渐缩小。

3. 实现测试自动化

有些测试(如回归测试和设置测试等)需要耗费大量时间,为了解决这个问题,就必须利用软件测试工具来缩短测试过程和时间,例如可以利用 GUI 的测试自动化软件来进行回归测试。

使用测试自动化工具的流程由设计测试用例、编写程序或脚本程序、执行测试、对比结果和生成报表、通报结果 5 部分组成。如果测试用例已生成,测试的执行可以由人工进行也可以利用测试工具自动执行。但设计的出发点是其能够被自动化执行。编写测试程序的目的是让所设计的测试用例能够自动执行,节省测试人员的时间。测试执行后的结果既可以自行对比是否有错,也可以记录程序本身所产生的错误。最后,所有的测试结果应该以报表的形式提交给测试人员。

总之,在测试过程中要使用测试工具更要善用测试工具。通过使用测试工具,不仅可以提高测试效率,而且还可以发现人工测试中很难发现的缺陷,这对于保证软件质量非常重要。

9.2 测试工具的分类

软件测试工具是提高软件测试效率的重要手段,是软件理论和技术发展的重要标志,也是软件测试技术实用化的重要标志;软件测试工具伴随软件测试技术的发展而发展。目前,按用途不同可将软件测试工具分为下述类型。

1. 测试设计工具

测试设计工具的功能是准备测试输入或测试数据。测试设计工具主要包括逻辑设计工具和物理设计工具。逻辑设计工具涉及说明、接口或代码逻辑,又称之为测试用例生成器。物理设计工具操作已有的数据或产生测试数据。例如可以随机从数据库中抽取记录的工具就是物理设计工具。从说明中获取测试数据的工具就是逻辑设计工具。

2. 测试管理工具

测试管理工具是指帮助完成测试计划,跟踪测试运行结果等的工具。这类工具是有助于需求、设计、编码测试及缺陷跟踪的工具。一般而言,测试管理工具用于对测试过程、测试计划、测试用例、测试实施进行管理,还包括对缺陷的跟踪管理。

Mercury Interactive 公司的 TestDirector、Rational 公司的 Test Manager、Compuware 公司的 TrackRecord 等都是较典型的测试管理工具软件。

3. 静态分析工具

静态分析工具直接对代码进行分析,不需要运行代码,也不需要对代码编译链接而生成可执行文件。静态分析工具一般是对代码进行语法扫描,找出不符合编码规范的地方,

根据质量模型评价代码的质量,生成系统的调用关系图等。Telelogic 公司的 Logiscope 软件、PR 公司的 PRQA 软件、Reasoning 公司的 Illuma 软件等都是较典型的静态分析工具。

4. 动态分析工具

动态测试工具一般采用"插桩"的方式,向代码生成的可执行文件中插入一些监测代码,用来统计程序运行时的数据。其与静态分析工具最大的不同点就是动态分析工具要求被测系统实际运行。Compuware 公司的 DevPartner 软件、Rational 公司的 Purify 等产品都是较典型的动态分析工具。

5. 覆盖测试工具

覆盖测试工具主要用于单元测试中。例如,对于安全性要求高或与安全有关的系统,则要求的覆盖程度也较高。覆盖测试工具还可以度量设计层次结构,如调用树结构的覆盖率等。

6. 黑盒测试工具

黑盒测试工具主要是指功能测试工具和系统测试工具。黑盒测试工具的原理是利用脚本的录制和回放,模拟用户的操作,然后将被测系统的输出记录下来并与预先给定的标准结果比较。黑盒测试工具可以大大减轻黑盒测试的工作量,在迭代开发的过程中,能够很好地进行回归测试。

IBM Rational 的 TeamTest、Robot,Compuware 公司的 QACenter,MI 公司的 Win Runner 等工具是较典型的黑盒测试工具。MI 公司的 LoadRunner、IBM Rational 的 Quantify、Radview 公司的 WebLoad、Microsoft 公司的 WebStress 等工具是较典型的系统测试工具。

例如,LoadRunner 用来进行性能测试、压力测试、模拟多用户、定位性能瓶颈。其功能包括:创建虚拟用户、创建真实的负载、定位性能问题、重复测试保证系统发布的高性能、EJB 的测试、支持无线应用协议、支持媒体流应用、完整的企业应用环境的支持。使用 LoadRunner 完成测试一般分为 4 个步骤:虚拟用户产生器创建脚本、中央控制器来调度虚拟用户、运行脚本、分析测试结果。

7. 负载和性能测试工具

利用性能测试工具可以检测每个事件所需要的时间。例如,性能测试工具可以测定典型或负载条件下的响应时间。负载测试可以产生系统流量,这种类型的测试工具用于容量和压力测试。Radview 公司的 WebLoad、Microsoft 公司的 WebStress、MI 公司的 LoadRunner 等工具是专门用于性能测试的工具。

8. GUI 测试驱动和捕获/回放工具

GUI 测试工具可使测试自动化执行,然后将测试输出结果与期望输出进行比较。此类测试工具可在任何层次中执行测试:单元测试、集成测试、系统测试或验收测试。捕获回放工具是目前使用的测试工具中最流行的一种。Rational 公司的 TeamTest、Robot,Compuware 公司的 QACenter,MI 公司的 Win Runner 等是较典型的工具。

9. 基于故障的测试工具

首先给出软件的故障模型，在此故障模型下，给出基于该故障模型的软件测试工具。这是一种有发展前景的软件测试工具。随着人们对软件故障认识的不断深入，软件的故障模型也会越来越完备，并更加符合实际。基于故障的软件测试工具有下述需要研究的问题。

(1) 故障模型的准确程度。

(2) 测试的准确程度。

(3) 测试的自动化程度。

Reasoning 公司的C++ 产品 C-In-spector 是较典型的基于故障的测试工具。

10. 专用测试工具

专用测试工具是用于某一专门应用领域或某些特殊用途的测试工具，针对 Web 应用的有 Work2bench、Web Application Stress Tool(WAS)、MI 公司的 Astra 系列；数据库测试工具 TestBytes 以及嵌入式测试工具 Test RealTime、CodeTest 等。

11. 测试辅助工具

测试辅助工具能够帮助测试人员更有效地进行测试，与测试过程相关。如 SmartDraw 用于绘制 UCML，进行负载压力测试需求分析，有助于测试前的准备工作；SDemo 将测试过程中的操作录制成可执行文件，并回放出来，可以避免随机出现的一些错误。

9.3　典型的软件测试工具

9.3.1　Logiscope 质量分析和测试工具

Logiscope 工具专用于软件质量保证和软件测试，主要功能如下。

- 在设计和开发阶段，可以对软件的体系结构和编码进行确认，可以尽早地检测关键部分，寻找潜在的错误。在构造软件时，定义测试策略。可帮助编制符合企业标准的文档，改进开发组之间的交流。
- 在测试阶段，可针对软件结构，度量测试覆盖的完整性，评估测试效率，确保满足要求的测试等级。特别是，Logiscope 还可以自动生成相应的测试分析报告。
- 在软件的维护阶段，验证已有的软件质量是否已得到保证，对于状态不确定的软件，可以迅速提交软件质量的评估报告，避免非受控修改引发的错误。

Logiscope 具有 3 个主要的功能：静态分析功能、语法规则分析功能、动态测试功能。

1. 静态分析功能

Logiscope 的质量模型描述了从质量方法学引入的质量因素、质量准则和质量度量元。即本模型是一个三层的结构组织：质量因素、质量准则、质量度量元。质量因素从用户角度出发，对软件的质量特性进行总体评估；质量准则从软件设计者角度出发，设计为

保障质量因素所必须遵循的法则；质量度量元从软件测试者角度出发，验证是否遵循质量准则。一个质量因素由一组质量准则来评估；一个质量准则由一组质量度量元来验证。其关系如图 9-1 所示。

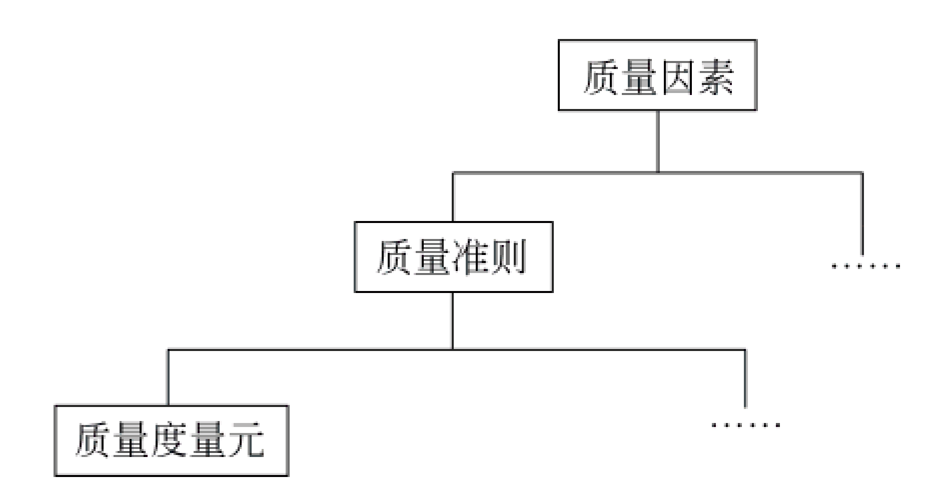


图 9-1 质量模型三层结构组织的关系

Logiscope 从系统、类和函数 3 个层次详细规定了上述质量特性及其组成关系。

（1）质量因素包括以下两方面。

① 可维护性。

② 可重用性。

（2）质量准则包括以下 4 个方面。

① 可分析性。

② 可修改性。

③ 稳定性。

④ 可测试性。

质量度量元较多，在此不详细描述。静态分析 Audit 部件将软件与所选的质量模型进行比较，生成软件质量分析报告。显示软件质量等级的概要图形，因此可以把精力集中在需要修改的代码部分。对度量元素和质量模型不一致的地方做出解释并提出纠正的方法。通过对软件质量进行评估及生成控制流图和调用图，发现最大可能发生错误的部分。一旦发现这些部分，可以使用度量元及控制流图、调用图等手段做进一步分析。

2. 语法规则分析功能

Logiscope 提供编码规则与命名检验，这些规则是根据业界标准和经验所制定的。因此可建立企业共同遵循的规则与标准，从而避免不良的编程习惯及彼此不相容的困扰。同时 Logiscope 还提供规则的裁剪和编辑功能，可以用 TCL、脚本和编程语言定义新的规则。

3. 动态测试功能

为控制测试的有效性，必须定义准则和策略以判断何时结束测试工作。准则必须是客观和可量化的元素。Logiscope 推荐对指令、逻辑路径和调用路径覆盖测试。根据应用的准则和项目相关的约束，可以定义使用的度量方法和要达到的覆盖率，度量测试的有效性。

Logiscope 也支持嵌入式软件的测试。嵌入式软件的测试最为困难，这是因为它的开发是用交叉编译方式进行的。在目标机上，不可能有多余的空间记录测试的信息，所以必须实时地将测试信息传到宿主机上，并实时在线显示。因此，对源代码的插装和目标机上的信息收集与回传成为问题的关键。Logiscope 很好地解决了这些技术，它支持各种实时操作系统上的应用程序的测试，也支持逻辑系统的测试。Logiscope 提供 VxWorks、pSOS、VRTX 等实时操作系统的测试库。

9.3.2 Rational Purify 测试自动化工具

测试自动化工具 Rational Purify 是 Rational PurifyPlus 工具中的一种，Purify 是一个面向 VC、VB 或者 Java 开发的程序，测试 Visual C/C++ 和 Java 程序代码中与内存有关的错误，确保整个应用程序的质量和可靠性的测试工具软件。在查找典型的 Visual C/C++ 程序中的传统内存访问错误，以及 Java 代码中与垃圾内存收集相关的错误方面，Rational Purify 能力优异。

对于 Visual C/C++ 开发者和测试者来说，运行时内存错误和泄漏是最重要也是最难寻找和修正的代码错误。

1. Rational Purify 的功能

(1) 对 VC 程序进行快捷、易于理解的错误检测。

(2) 即使没有代码也可以实现错误检查。

(3) 代码覆盖寻找未测试代码部分。

Purify 可以和 Microsoft Visual Studio 自动集成并且无需特殊的编译。可以在不改变工作方式的前提下使用 Purify。

对于 Java 的开发者和测试者来说，程序员和测试人员可以将 Rational Purify 和所支持的 JVM(Java 虚拟机)相结合，以改善和优化 Java Applet 和应用程序的内存功效。Purify 提供了一套功能强大的内存使用状况分析工具，可以找出消耗了过量内存或者保留了不必要对象引用的方法调用。

2. Rational Purify 可检查的错误类型

Rational Purify 的学习和使用过程都非常简单。能够协助快速找出编程错误。Rational Purify 可以自动找出错误的准确位置。如果有源代码，则可以从 Rational Purify 中启动相应的编辑器，从而快速修复错误。使用 Rational Purify 特有的功能 PowerCheck，可以按模块逐个调整所需的检查级别。这样就可以把精力集中在最重要的代码上，简单选择最小或准确即可。最小检查可以快速查出常见的运行写入错误和 Windows API 错误；对于关键模块，准确检查将用行业强度检查来查找内存访问错误，这样就可以在最重要的代码块上确定调试的优先级并更有效地工作。对于同时进行代码覆盖分析的测试，选择覆盖级别，如代码行或函数，以便更好地控制错误检查和数据覆盖。Rational Purify 可检查的错误类型有以下一些。

(1) 堆栈相关错误。

(2) 垃圾内存收集——Java 代码中相关的内存管理问题。

(3) COM 相关错误。

(4) 指针错误。

(5) 内存使用错误。

(6) Windows API 相关错误。

(7) Windows API 函数参数错误和返回值错误。

(8) 句柄错误。

3. Rational Purify 可检测错误的代码

(1) ActiveX(OLE/OCX)控件。

(2) COM 对象。

(3) ODBC 构件。

(4) Java 构件、Applet、类文件、JAR 文件。

(5) Visual C/C++ 源代码。

(6) Visual Basic 应用程序内嵌的 Visual C/C++ 构件。

(7) 第三方和系统 DLL。

(8) 支持 COM 调用的应用程序中的所有 Visual C/C++ 构件。

4. Rational Purify 的测试信息色彩

(1) 红色代表内存块没有被分配和初始化。

(2) 蓝色代表内存块已经被分配并且初始化了。

(3) 黄色代表内存块已经被分配但是没有初始化。

5. Rational Purify 的测试信息名称缩写

(1) Array Bounds Read(ABR):数组越界。

(2) Beyond Stack Read(BSR):堆栈越界。

(3) Free Memory Read(FMR):空闲内存访问。

(4) Invalid Pointer Read(IPR):非法指针访问。

(5) Null Pointer Read(NPR):空指针访问。

(6) Uninitialized Memory Read(UMR):未初始化内存访问。

9.3.3 Win Runner 功能测试工具

企业级应用是指 Web 应用系统、ERP 系统、CRM 系统等。这些应用系统在发布之前、升级之后都要经过测试,确保所有功能都能正常运行,没有任何错误。如何有效地测试不断升级更新且在不同环境下运行的应用系统,是更需要面临的问题。

如果时间或资源有限,这个问题将更为突出。人工测试的工作量太大,还要额外的时间来培训新的测试人员等。为了确保那些复杂的企业级应用在不同环境下都能正常可靠地运行,就得需要一个能简单操作的测试工具来自动完成应用程序的功能性测试。

Win Runner 是一种企业级的功能测试工具,用于检测应用程序是否能够达到预期的功能以及能否正常运行。通过自动录制、检测和回放用户的应用操作,Win Runner 能够有效地帮助测试人员对复杂的企业级应用的不同发布版本进行测试,提高测试人员的工作效率和质量,确保跨平台的、复杂的企业级应用无故障发布及长期稳定运行。

1. Win Runner 的主要功能

1) 轻松创建测试

Win Runner 提供了两种测试创建方式,以满足测试团队中业务用户和专业技术人员的不同需求。第一种方法只需操作鼠标和键盘,完成一个标准的业务操作流程,Win

Runner 自动记录测试人员的操作并生成所需的脚本代码。这样，可使业务用户也可以轻松创建完整的测试。另一种方法是通过直接修改测试脚本以满足各种复杂测试的需求。

2）插入检查点

在记录一个测试的过程中，可以插入检查点，检查在某个时刻或某个状态下，应用程序是否运行正常。在插入检查点后，Win Runner 会收集一套数据指标，在测试运行时对其一一验证。Win Runner 提供几种不同类型的检查点，包括文本的、GUI、位图和数据库。例如，用一个位图检查点就可检查公司的图标是否出现于指定位置。

3）检验数据

除了创建并运行测试，Win Runner 还能验证数据库的数值，从而确保业务交易的准确性。例如，在创建测试时，可以设定哪些数据库表和记录需要检测；在测试运行时，测试程序就会自动核对数据库内的实际数值和预期的数值。Win Runner 自动显示检测结果，并在有更新/删除/插入的记录上突出显示以引起测试人员的注意。

4）增强测试

为了彻底全面地测试一个应用程序，需要使用不同类型的数据来测试。Win Runner 的数据驱动向导（Data Driver Wizard）可以通过简单地单击几下鼠标，就把一个业务流程测试转化为数据驱动测试，从而反映多个用户各自独特且真实的行为。

2. Win Runner 的使用

1）运行测试

创建好测试脚本，并插入检查点和必要的添加功能后，就可以开始运行测试。运行测试时，Win Runner 自动操作应用程序，就像一个真实的用户根据业务流程执行软件中的每一步操作。测试运行过程中，如有网络消息窗口出现或其他意外事件出现，Win Runner 也会根据预先的设定排除这些干扰。

2）分析结果

测试运行结束后，需要分析测试结果。Win Runner 通过交互式的报告工具来提供详尽的、易读的报告。报告中会列出测试中发现的错误内容、位置、检查点和其他重要事件，帮助测试人员对测试结果进行分析。这些测试结果还可以通过 Mercury Interactive 的测试管理工具 Test Director 来查阅。

3）维护测试

随着时间的推移，开发人员会对应用程序做进一步的修改，并需要增加另外的测试。使用 Win Runner，不必对程序的每一次改动都重新创建测试。Win Runner 可以创建在整个应用程序生存周期内都可以重复使用的测试，从而大大地节省时间和资源，充分利用自己的测试投资。

每次记录测试时，Win Runner 会自动创建一个 GUI Map 文件以保存应用对象。这些对象分层次组织，既可以总揽所有的对象，也可以查询某个对象的详细信息。一般而言，对应用程序的任何改动都会影响到成百上千个测试。通过修改一个 GUI Map 文件而无需改动影响到的无数个测试，Win Runner 可以方便地实现测试重用。

4）帮助应用程序为无线应用做准备

随着无线设备种类和数量的增加，应用程序测试计划需要同时满足传统的基于浏览

器的用户和无线浏览设备，如移动电话、传呼机和个人数字助理（PDA）。无线应用协议是一种公开的、全球性的网络协议，用来支持标准数据格式化和无线设备信号的传输。

使用 Win Runner，测试人员可以利用微型浏览模拟器来记录业务流程操作，然后回放和检查这些业务流程功能的正确性。

9.3.4 TestDirector 测试管理系统

TestDirector 是第一个基于 Web 的测试管理系统，它可以在公司内部或外部进行全球范围内测试的管理。通过在一个整体的应用系统中集成测试管理的各个部分，主要包括需求管理、测试计划、测试执行以及错误跟踪等功能，应用 TestDirector 工具可以明显加速测试过程。

TestDirector 在消除组织机构间、地域间的障碍方面功能强大。它能让测试人员、开发人员或其他的 IT 人员通过一个中央数据仓库，在不同地方可以交互测试信息。TestDirector 将测试过程流水化：从测试需求管理，到测试计划，测试日程安排，测试执行到出错后的错误跟踪，仅仅在一个基于浏览器的应用中便可完成，而不需要每个客户端都安装一套客户端程序。TestDirector 的主要功能如下所述。

1. 需求管理

系统的需求推动整个测试过程。TestDirector 的 Web 界面简化了这些需求管理过程，以此可以验证应用软件的每一个特性或功能是否正常。通过提供一个比较直观的机制将需求和测试用例、测试结果和测试报告的错误联系起来，从而确保能达到最高的测试覆盖率。

将需求和测试联系起来有下述两种方式。

（1）TestDirector 捕获并跟踪所有首次发生的应用需求。测试人员可以在这些需求基础上生成一份测试计划，并将测试计划对应用户的需求。例如，假如有 25 个测试计划同时对应同一个应用需求。TestDirector 能方便地管理需求和测试计划之间可能存在的一种多对多的关系，确保每一个需求都经过测试。

（2）由于 Web 应用是不断更新和变化的，需求管理允许测试人员修改需求，并确定目前的应用需求已拥有了一定的测试覆盖率。它们帮助决定一个应用软件的哪些部分需要测试，哪些测试需要开发，完成的应用软件是否满足了用户的要求。对于任何动态地改变 Web 应用，必须审阅测试计划是否准确，确保测试计划符合当前最新的应用要求。

2. 测试计划

制定测试计划是测试过程中重要的环节。它为整个测试提供了一个结构框架。TestDirector 的 Test Plan Manager 在测试计划期间，为测试小组提供一个关键要点和 Web 界面来协调团队间的沟通。

Test Plan Manager 指导测试人员如何将应用需求转化为具体的测试计划。能帮助测试人员规划如何测试待测的应用软件，从而使测试人员能组织起明确的任务和责任。Test Plan Manager 可用多种方式来建立完整的测试计划。可以从草图上建立一份计划，或根据测试人员用 Requirements Manager 所定义的应用需求，通过 Test Plan Wizard 快

捷地生成一份测试计划。如果已经将计划信息以文档形式储存,可以再利用这些信息,并将它导入到 Test Plan Manager。它把各种类型的测试汇总在一个可折叠式目录树内,测试人员可以在一个目录下查询到所有的测试计划。Test Plan Manager 还能帮助测试人员完善测试设计和以文件形式描述测试步骤,包括对每一项测试,用户反应的顺序,检查点和预期的结果。TestDirector 还能为每一项测试添加附属文件,如 Word、Excel、HTML,用于更详尽地记录每次测试计划。

当测试人员需要频繁更新测试计划、优化测试内容时,TestDirector 仍能简单地将应用需求与相关的测试对应起来。TestDirector 还可支持不同的测试方式来适应特殊的测试流程。

多数的测试项目需要人工和自动测试的结合,包括健全和还原系统测试。但即使符合自动测试要求的工具,在大部分情况下也需要人工的操作。启用一个自动化切换机制,能让测试人员决定哪些重复的人工测试可转变为自动脚本以提高测试速度。TestDirector 还能简化将人工测试切换到自动测试脚本的转化,并可立即启动测试设计过程。

3. 安排和执行测试

当测试计划建立后,TestDirector 的测试实验室管理为测试日程的制定提供一个基于 Web 的框架。它的 Smart Scheduler 功能根据测试计划中创立的指标对运行着的测试执行监控。

当网络上的任何一台主机空闲,测试就可以在上面执行。Smart Scheduler 能自动分辨是系统错误还是应用错误,然后将测试重新安排到网络上的其他机器。

对于不断改变的 Web 应用,经常性地执行测试对于追查错误发生的环节和评估应用质量至关重要。然而,这些测试的运行都要消耗测试资源和时间。使用 Graphic Designer 图表设计,测试人员可以很快地将测试分类以满足不同的测试目的,如功能性测试、负载测试、完整性测试等。它的拖动功能可简化设计和排列在多个机器上运行的测试,最终根据设定好的时间、路径或其他测试的成功与否,为序列测试制定执行日程。Smart Scheduler 可以在更短的时间内,在更少的机器上完成更多的测试。

用 Win Runner、Astra QuickTest、Astra LoadTest 或 LoadRunner 来自动运行功能测试或负载测试,无论成功与否,测试信息都会被自动汇集传送到 TestDirector 的数据储存中心。同样,人工测试也以此方式运行。

4. 缺陷管理

测试完成后,当有错误发现时,还可指定相关人员及时纠正。

TestDirector 的错误管理贯穿于测试的全过程,以提供管理系统在终端间的错误跟踪,从最初的问题发现到修改错误再到检验修改结果。由于同一项目组中的成员经常分布于不同的地方,TestDirector 基于浏览器的特征,使错误管理能让多个用户在任何地方都可通过 Web 查询错误跟踪情况。利用错误管理机制,测试人员只需进入一个 URL,就可汇报和更新错误,过滤整理错误列表并作趋势分析。在进入一个错误案例前,测试人员还可自动执行一次错误数据库的搜寻,确定是否已有类似的案例记录,从而可以避免重复

劳动。

5. 图形化和报表输出

测试过程的最后一步是分析测试结果，确定应用软件是否已经部署成功或需要再次运行的测试。TestDirector 常规化的图表和报告在测试的任一环节都可以帮助测试人员对数据信息进行分析。

TestDirector 还以标准的 HTML 或 Word 形式提供一种生成和发送正式测试报告的一种简单方式。测试分析数据可简便地输入到一种工业标准化的报告工具，如 Excel、ReportSmith、CrystalReports 和其他类型的第三方工具。

9.4 测试工具的选择

在选择测试工具时，可从以下几个方面考虑。

1. 功能

功能是最受关注的内容，选择一个测试工具首先就是看它的功能。当然，这并不是说测试工具提供的功能越多越好，在实际的选择过程中，适用性才是根本，为不需要的功能花费费用是不明智的行为。目前，同类的软件测试工具之间的基本功能大同小异，各种软件的功能也大致相同，只是侧重点不同。例如，同为白盒测试工具的 Logiscope 和 PRQA 软件，它们提供的基本功能大致相同，只是在编码规则、编码规则的定制及所用的代码质量标准不同。

除了基本的功能之外，下述功能也可以作为参考。

(1) 报表功能：测试工具生成的结果最终要由人进行解释，而且，查看最终报告的人员不一定对测试很熟悉，因此，测试工具能否生成结果报表及能以什么形式提供报表是需要考虑的因素。

(2) 集成能力：测试工具的集成能力也是必须考虑的因素，集成包括两个方面的意思。首先，测试工具能否和开发工具进行良好的集成；其次，测试工具能否和其他测试工具进行良好的集成。

(3) 操作系统和开发工具的兼容性：测试工具可否跨平台，是否适用于目前使用的开发工具，这些问题也是在选择一个测试工具时必须考虑的问题。

2. 价格

除了功能之外，价格就应该是最重要的因素了。测试工具的价格应该在承受范围之内，既不能为了省钱买测试功能有限的产品，也不能不顾承受能力去购买价格昂贵的产品，购买性价比最高的产品才是最明智的选择。

3. 选择适合于软件生存周期各阶段的工具

测试的种类随着测试所处的生存周期阶段的不同而不同，因此为软件生存周期选择其所使用的恰当工具就非常必要。如：程序编码阶段可选择 Telelogic 公司的 Logiscope 软件、Rational 公司的 Purify 系列等；测试和维护阶段可选择 Compuware 公司的

DevPartner 和 Telelogic 公司的 Logiscope 等。

4. 连续性

测试工具是测试自动化的一个重要步骤之一，在引入/选择测试工具时，必须考虑测试工具引入的连续性。也就是说，对测试工具的选择必须有一个全面的考虑，分阶段、逐步地引入测试工具。

使用了测试工具，并不是说已经进行了有效测试，测试工具通常只支持某些应用的测试自动化，因此在进行软件测试时常用的做法是：使用一种主要的测试自动化工具，然后用传统的编程语言如：Java、C++、Visual Basic 等编写测试自动化脚本以弥补测试工具的不足。

9.5　测试工具的局限性

(1) 软件测试工具不能取代手工测试，手工测试可以比自动测试发现更多的缺陷。

在实际中，不可能将所有测试活动进行自动化。有些测试使用手工测试比自动测试要简单，下列情况就不适合测试自动化。

① 测试很少运行。如一年只运行一次，这种情况就不适合自动化。

② 软件不稳定。如软件版本更新后，由于用户界面和功能变化很大，造成修改相应的测试自动化的开销较大。

③ 涉及物理交互的测试。如在读卡机上划卡、断开某些设备的连接、开/关电源等。

(2) 测试工具对期望结果的正确性依赖性极大。

测试在首次运行时最有可能发现缺陷。如果测试已经运行并通过，再运行相同的测试发现新缺陷的可能性则小得多。除非测试正执行一段已修改过的代码，或者由于软件其他部分的修改影响到该代码，或者在不同的环境中运行。但是，测试工具没有发现任何缺陷并不意味该软件没有缺陷。如果测试本身就有缺陷或者期望的输出不正确，而测试工具只能简单地判断实际结果和期望结果之间的区别。

(3) 测试自动化可能会制约软件开发。

由于测试工具自身也是软件，因此测试工具与其他软件的互操作性和兼容性是一个值得关注的问题。如果软件在设计和使用时可测试性考虑得不够充分，也将给自动测试带来较大的难度。

(4) 测试工具本身不具有想象力。

测试工具也是软件，只是按照指令执行。测试工具和测试者都可以按指令执行一组测试，但人工测试可以具有智能性，可用不同的方式完成相同的任务。

小　　结

应用测试工具的目的是提高测试效率、减少测试过程中的重复劳动和实现测试工作的自动化，并且可能会发现在手工测试中很难发现的缺陷，所以说应用测试工具对保证软

件质量和做好测试工作非常重要。

根据不同的测试原理，可以将测试工具分为白盒测试工具、黑盒测试工具、性能测试工具、测试管理工具等几个大类。

本章还介绍了几种比较典型的测试工具：用于静态测试的 Logiscope、用于动态测试的 Rational Purify、企业级的功能测试工具 Win Runner 和测试管理工具 TestDirector。在最后介绍了在选择测试工具时应注意的几个方面。

通过本章内容的学习，可以对测试工具的作用、应用范围、分类、典型软件测试工具有较全面的理解，为合理地应用测试工具，开发高质量软件建立了坚实基础。

习 题 9

1. 使用测试工具的目的。
2. 选择测试工具时的注意事项。
3. 选择不同的测试工具去测试自己编写的程序，并生成文档报告。
4. 测试自动化工具的流程由________、________、________、________和________5 部分组成。
5. 练习使用其他免费使用、目的明确的测试工具。

参 考 文 献

[1] 古乐，史九林. 软件测试技术概论. 北京：清华大学出版社，2004.
[2] 柳纯录. 软件评测师教程. 北京：清华大学出版社，2005.
[3] 路晓丽，葛玮，袭晓庆等. 软件测试技术. 北京：机械工业出版社，2009.
[4] 金茂忠. 信息系统测试. 北京：中央广播电视大学出版社，2005.
[5] Paul C. Jorgensen. 软件测试. 韩柯，杜旭涛译. 北京：机械工业出版社，2007.
[6] Cem Kaner，Jack Falk，Hung QuocNguyen. 计算机软件测试. 王峰，陈杰，喻琳译. 北京：机械工业出版社，中信出版社，2004.
[7] Srinivasan Desikan，Gopalaswamy Ramesh. 软件测试原理与实践. 韩珂，李娜等译. 北京：机械工业出版社，2009.
[8] Ron Patton. 软件测试. 张小松，王钰，曹跃译. 北京：机械工业出版社，2007.
[9] 覃征，邢剑宽，董金春等. 软件体系结构. 北京：清华大学出版社，2008.
[10] 陈明. 软件工程导论. 北京：机械工业出版社，2010.